Universidade Federal de Pernambuco
Graduação em Ciência da Computação
Centro de Informática

Trabalho de Graduação

Uma Especificação de Desenvolvimento de Serviços para Televisão Digital Interativa

Andrino Soares de Souza Coêlho

Orientador: Carlos André Guimarães Ferraz

Recife,
22 de Agosto 2005.

Universidade Federal de Pernambuco
Graduação em Ciência da Computação
Centro de Informática

Trabalho de Graduação

Uma Especificação de Desenvolvimento de Serviços para Televisão Digital Interativa

Andrino Soares de Souza Coêlho

A Televisão Digital Interativa está cada vez mais próxima da realidade, inclusive no Brasil. Com seu advento, além de assistir a programação de TV, o telespectador poderá interagir, em tempo real, com os mais diversos serviços e aplicações.
Esse trabalho de graduação foca-se na especificação de tais serviços. Seu objetivo maior é padronizar o processo de elaboração e desenvolvimento destes.

Orientador: Carlos André Guimarães Ferraz

Recife,
22 de Agosto 2005.

Coelho, Andrino S. S.

Uma Especificação de Desenvolvimento de Serviços para Televisão Digital Interativa/ Andrino S. S. Coelho. Recife, 2005.

Trabalho de Graduação – Universidade Federal de Pernambuco
Centro de Informática

Orientador: Ferraz, Carlos A. Guimarães

Coloco em cada caso um princípio, aquele que julgo o mais sólido, e tudo o que parece estar em consonância com ele - quer se trate de causas ou de qualquer outra coisa - admito como verdadeiro, admitindo como falso o que com ele não concorda.

...

Admitamos pois - o que me servirá de ponto de partida e de base – que existe um Belo em si e por si, um Bom, um Grande, e assim por diante. Se admitires a existência dessas coisas, se concordares comigo, esperarei que elas me permitirão tornar-te clara a causa, que assim descobrirás, que faz com que a alma seja imortal.

- Sócrates

Aos Meus Pais

Agradecimentos

"The thread of our life would be dark, heaven knows! If it were not with friendship and love interweaved."

- Thomas Moore

Aos meus pais, patronos e incentivadores de minha formação.

Aos meus familiares e amigos, pelo suporte e compreensão.

Ao meu orientador e, sempre amigo, Carlos Ferraz, pelos fundamentais e duradouros ensinamentos.

Aos meus amigos de curso, em especial Hermano, Jayro, Rafael e Thiago, pelos verões e invernos vividos em companheirismo.

Aos meus companheiros do projeto C.E:N.A.S., pelas calorosas e enriquecedoras discussões.

Aos meus companheiros e, acima de tudo amigos, do C.E.S.A.R., em especial Diogo, Paulyne e Tarcísio, pelos conhecimentos partilhados.

Aos meus mestres do CIN, em especial o Prof. Geber, pelo conhecimento adquirido.

E a todos que contribuíram, direta ou indiretamente, para a conclusão deste trabalho, meus sinceros agradecimentos.

RESUMO

O futuro da Televisão é digital e, repleto de interatividade. O telespectador receberá um sinal com qualidades de imagem e áudio muito superior aos atuais. Entretanto, a maior contribuição desta evolução natural, se dará através da interatividade, que promoverá a utópica inclusão social. Este processo competirá à chamada *"computacionalização"* da TV, sendo através desta que um usuário, o qual nunca tenha acessado à Internet ou lido um E-mail os farão.

Neste cenário de vanguarda, é imprescindível a realização de uma profunda análise sobre serviços interativos, à luz dos meios de comunicação. O que são? Como concebê-los? Elaborá-los? Desenvolvê-los? Estes são os focos principais deste trabalho, almejando uma introdução ao *middleware* da televisão digital interativa, apresentando os serviços e as aplicações suportadas, bem como uma proposta de um *Gerenciador de Aplicações*.

Palavras-chave: televisão digital, interatividade, *middleware*, serviços, aplicações, Gerenciador de Aplicações.

Abstract

The Television´s future is digital and full of interactivity. The televiewer will receive a tv signal with high quality video and audio. Besides, the natural evolution´s greater contribution, will be the interactivity, that will promote the utopia of social inclusion. This process will provide computing capability to a TV set, allowing people who have never accessed internet and e-mail to do it.

In this vanguard scenario, a deep analysis about what interactive services are needed, considering the current communication technology. What are they? How to conceive them? How to design and develop them? This is the main subject of this work. It will also be shown an introduction to the interactive digital television´s middleware, presenting supporting services and applications, as well as an Application Manager proposal.

Keywords: digital television, interactive, middleware, services, applications, application manager

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo I

Introdução

"O único lugar aonde o sucesso vem antes do trabalho é no dicionário."

- Albert Einstein

Vivemos atualmente um momento de transição entre a era da televisão analógica e um futuro digital repleto de interatividade, onde a televisão dará acesso aos mais diversos serviços, indo muito além da recepção de conteúdo audiovisual. Este novo cenário só será possível apoiado na evolução constante das técnicas de codificação digital de áudio e vídeo, aliada aos novos esquemas eficientes de modulação para transmissões digitais[01]. Um telespectador poderá interagir, por exemplo, com um comercial de automóveis, analisando preços, modelos, dispositivos opcionais e, até mesmo, agendar um *"test-drive"* na concessionária mais próxima de sua residência.

Estas características podem ser resumidas à "agregar capacidade computacional à TV"[01]. Computação, esta, contida em um dispositivo tecnológico conectado à mesma, o qual realizará o tratamento correto do sinal de radiodifusão, bem como, decodificação e exibição de uma maneira consistente da programação de televisão, das aplicações e dos serviços avançados. A este dispositivo dá-se o nome de *"set-top box"* ou, simplesmente, terminal de acesso.

No terminal de acesso encontram-se tanto o *hardware* como o *software* requerido para prover a interação do usuário com os serviços providos pelas *Geradoras de Conteúdo* e, retransmitidos por uma emissora. Sendo, portanto, necessário compartilhar um protocolo padronizado entre o terminal de acesso e os equipamentos desta última.

Os dados são enviados ou recebidos pelo terminal de acesso, através de um canal unidirecional de difusão ou por um canal bidirecional interativo. A interação com o usuário final ocorre por dispositivos de entrada - tais como, controle remoto e teclado, e dispositivos de saída – tais como, aparelhos de TV analógico ou digital, monitores de computador e sistemas de som.

O *hardware* do terminal de acesso deve ser enxuto, tendo como principais requisitos: capacidades de armazenamento e de processamento, bem como, acesso à rede. Um de seus

principais componentes é um demultiplexador[04], responsável pela separação do conteúdo de vídeo, áudio e dados.

A camada de *software* inclui, dentre outros, um *Real-time Operating System* e um mecanismo de interatividade. Deve-se destacar, no entanto, o *middleware* – vide Ilustração 1 –, já que o mesmo é o neologismo criado para designar camadas de software que não constituem diretamente aplicações, mas que facilitam o uso de ambientes ricos em tecnologia da informação. A camada de *middleware* concentra serviços como identificação, autenticação, autorização, diretórios, certificados digitais e outras ferramentas para segurança .

No contexto de TV Digital Interativa é o software que controla suas principais facilidades (grade de programação, menus de opção), inclusive a possibilidade de execução de aplicações (interatividade)[19]. É o elemento capaz de fornecer uma abstração do sistema, denominada como *Application Programming Interface (API)*, para as aplicações e os usuários, escondendo toda a complexidade dos mecanismos definidos pelos padrões, protocolos de comunicação e até mesmo sistema operacional do equipamento[01].

Este *middleware* é composto por diversos módulos, como: processamento de dados, apresentação de mídia, gerenciador de eventos do usuário, acesso condicional etc. Dentre estes, um dos mais importantes é o *Gerenciador de Aplicações*, o qual será um dos focos deste trabalho.

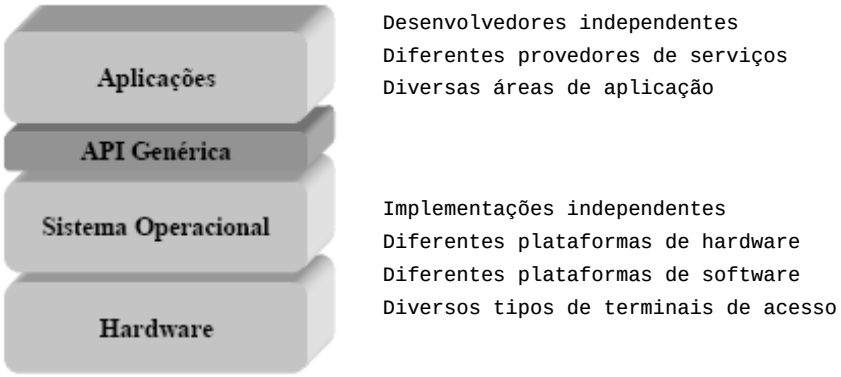

Ilustração 1 - Plataforma do Terminal de Acesso

1.1. SERVIÇOS

O serviço é a principal unidade de produção e consumo na TV Digital. É constituído de partes menores, partindo da atomicidade do evento – agrupamento de *streams* elementares (A/V/D) com tempo definido de início e fim, passando pelos programas – concatenação de um ou mais eventos produzidos por um estúdio, e, alcançando a programação – seqüência de programas controlada por um difusor, veiculada em uma determinada faixa de horário.

Programação, em TV Digital, é sinônima de serviço. Há ainda, uma unidade composta por diversos serviços, denominada de *bouquet* - unidade de distribuição das programações de uma Central de Produções, a qual pode ser composta por serviços (programação) produzidos por vários estúdios[04].

A *JavaTV API* [05] define serviço como uma abstração que provê maneiras comuns de se referir a uma grande gama de conteúdo em um ambiente de *broadcast*[06]. Este conteúdo é formado por seqüências elementares de bits compartilhando a mesma base de tempo. Estas seqüências são então multiplexadas entre si (primeiro nível de multiplexação), para formarem um programa – equivalente a um canal na TV tradicional[04].

Estes programas são então multiplexados entre si (segundo nível de multiplexação), formando a seqüência de transporte do sistema. Isto provê uma funcionalidade interessante, já que permite livres combinações de seqüências *PES*[1] *(Packetized Elementary Stream)*, incluindo repetições e seleções de seqüências específicas. Este sistema de multiplexação pode ser visto na Ilustração 2.

É, então, enviado um sinal – formato *MPEG-2*[12], contendo este *stream*. O terminal de acesso o capta, realizando a demultiplexação do conteúdo. Com a partição efetuada e as descrições do serviço devidamente alocadas em tabelas, o terminal disponibiliza as informações para controle do *middleware*.

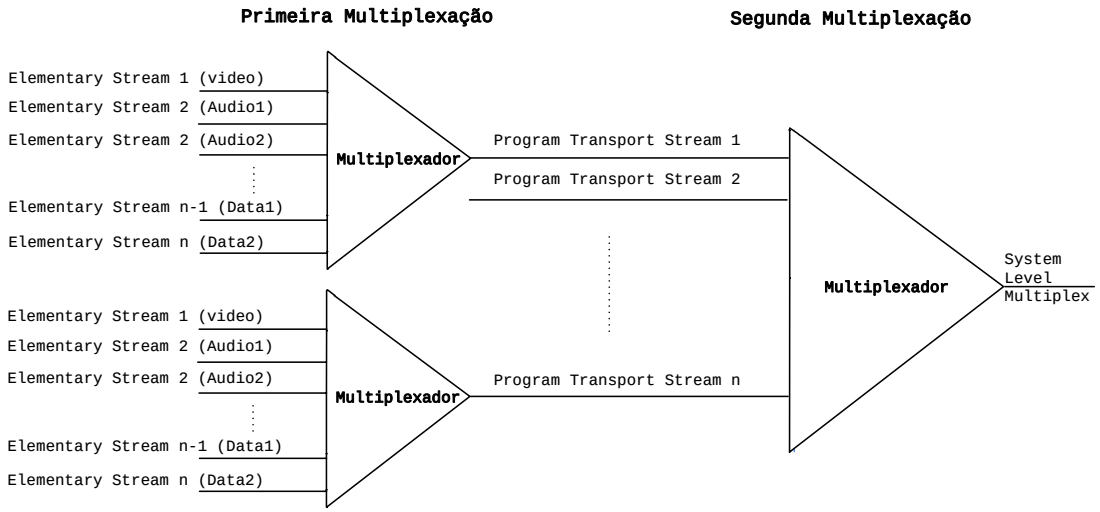

Ilustração 2 - Sistema de Multiplexação

[1] As principais finalidades da segmentação PES são viabilizar a sincronização das seqüências elementares de bits de um mesmo programa. As seqüências de áudio e vídeo passam obrigatoriamente por essa etapa. O processo de geração de segmentos PES pode ser realizado diretamente pelo subsistema de multiplexação e transporte ou pelo próprio codificador da aplicação geradora da seqüência elementar de bits[04].

O *stream* elementar de dados pode encapsular aplicações. Estas serão decodificadas no *middleware*, tendo seu ciclo de vida gerenciado pelo módulo ***Gerenciador de Aplicações***.

1.2. Estado da Arte

Existem atualmente alguns padrões de TV Digital, dos quais há de se destacar o *ATSC-DASE*, *DVB-MHP*, *ISDB-ARIB* e o *GEM*. Neles, todavia, há nuances quanto à definição de como se desenvolver serviços. Nesta seção apresenta-se uma breve descrição sobre cada padrão, sendo arrematada por uma comparação breve entre os mesmos.

1.2.1. ATSC-DASE

O *Advanced Television Systems Comitee (ATSC)*[10] é desenvolvido por uma organização dos Estados Unidos que desenvolveu padrões técnicos de transmissão terrestre (ATSC-T) e via cabo (ATSC-C). Tendo sido também definido um *middleware* para a plataforma de receptores, o *DTV-Application*[11][07]. A Ilustração 3 apresenta a arquitetura deste padrão entre as opções atuais para a mesma.

O interesse principal em seu desenvolvimento era a transmissão de serviços de TV em alta definição, mais conhecido como *HDTV (High Definition Television)*. O *HDTV* permite uma maior definição de vídeo – formato *widescreen*, ou seja, telas com aspecto 16:9, similares às do cinema – e, fidelidade no áudio, chegando a 6-1 canais de distribuição do som.

Todavia existem aspectos, nos quais este padrão é pobre: no suporte de serviços interativos e na transmissão para dispositivos móveis. Esta última característica é facilmente entendida ao se analisar a malha de cabeamento nos Estados Unidos, mostrando o porquê de um maior investimento neste tipo de retransmissão.

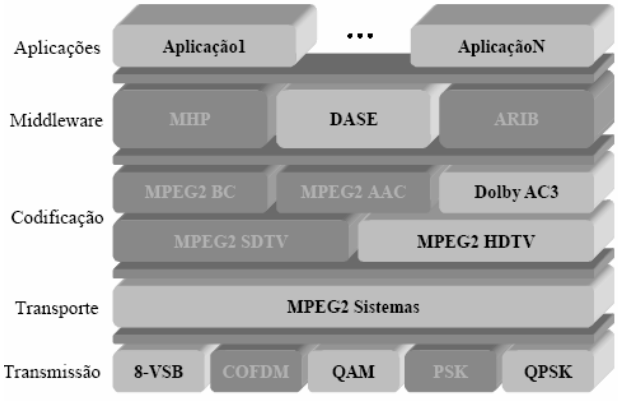

Ilustração 3 - Arquitetura do ATSC

1.2.2. DVB-MHP

Fruto do consórcio europeu *The Digital Vídeo Broadcasting Project (DVB)*[08], constituído de entidades de diversas áreas da indústria televisiva, tais como: emissoras, fabricantes de equipamentos, desenvolvedores de softwares, geradoras de conteúdo e órgãos reguladores[07]. Durante seu desenvolvimento foram criados padrões técnicos para transmissão terrestre *(DVB-T)*, transmissão via satélite *(DVB-S)* e via cabo *(DVB-C)*, assim como um *middleware* para a plataforma de receptores, o *Multimedia Home Plataform (MHP)*[09]. A Ilustração 4 apresenta a arquitetura deste padrão entre as opções atuais para a mesma.

O *middleware* do *DVB*[08] é dotado de uma especificação para desenvolvimento de serviços interativos (aplicações), contudo ainda há muita coisa a ser detalhada, principalmente no que se diz respeito à obtenção de dados do sistema e da interação interaplicações – podendo, esta, ser *in-loco* como remota.

O padrão *DVB* é o de mais fácil adaptação – já que possui uma maior documentação associada, além de contar com um grande *lobby* dos países europeus, o que o levou a ser, atualmente, o padrão mais adotado mundialmente.

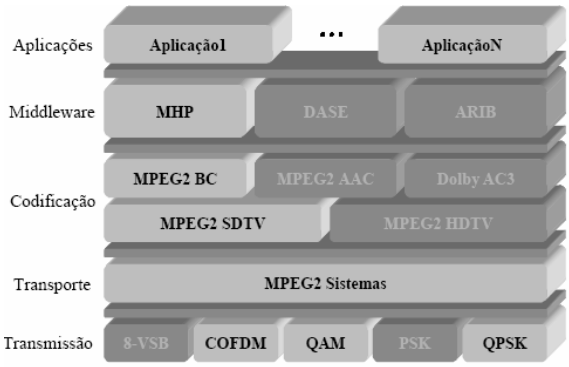

Ilustração 4 - Arquitetura do DVB

1.2.3. ISDB-ARIB

Concebido, produzido e adotado exclusivamente no Japão, através do grupo *Japanese Digital Broadcasting Experts Group (DiBEG´s)*. Conta com as melhores características dos outros padrões, até por ter sido elaborado à luz dos mesmos. Dá suporte técnico a transmissões no âmbito terrestre (*ISDB-T* e *ISDB-TSB*, ou *Terrestrial Sound Broadcasting*), via satélite *(ISDB-S)* e via cabo *(ISDB-C)*[07]. Sua adaptação torna-se complicada devido à fronteiras com a língua e escassa documentação de apoio. A Ilustração 5 apresenta a arquitetura deste padrão entre as opções atuais para a mesma.

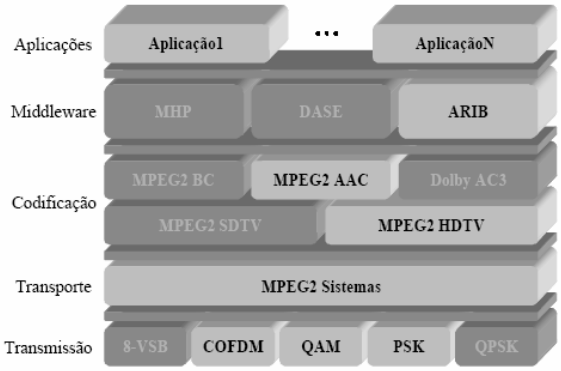

Ilustração 5 - Arquitetura do ISDB

1.2.4. GEM

O *Globally Executable MHP (GEM)* é uma tentativa de harmonização dos *middlewares* existentes no mundo. O *GEM* traz em sua especificação um subconjunto do MHP, com funcionalidades gerais que devem ser implementadas por todo e qualquer *middleware*. Tanto os Estados Unidos como o Japão já adotaram o *GEM* - vide Ilustração 6, bem como o Brasil em seus estudos. As metas definidas pelo *DVB* para o *GEM* são:

- Maximizar a interoperabilidade entre as especificações baseadas no *GEM* de diferentes organizações;
- Maximizar a presença de componentes *MHP* no *middleware*, perfazendo uma economia na cadeia de *interactive broadcast*;
- Facilitar o desenvolvimento de aplicações portáveis aos diferentes *middlewares*.

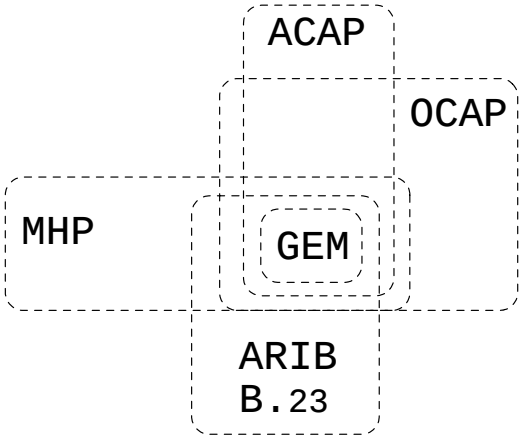

Ilustração 6 - GEM e Interação com outros Middlewares

1.2.5. Comparação Entre Padrões

O Brasil em sua busca por um padrão de TV Digital Interativa realizou através da ANATEL uma comparação entre os padrões. Os dados encontram-se resumidos na Tabela 1 -

Padrões de TV Digital – é necessário estar ciente de que os dados são da primeira pesquisa realizada.

Com a conclusão da pesquisa a ANATEL recomendou o *ISDB* – já que o mesmo sofre menos interferência que os demais padrões, além de que seu canal de transmissão pode ser subdivido em vários sub-canais, permitindo assim a transmissão simultânea de vários serviços. O *DVB* foi referendado, dada sua fácil adaptação e atual nível de desenvolvimento e adoção no mercado, sendo o *ATSC* [10] descartado como o menos recomendado.

Padrões	*Middleware*	Sist. Vídeo	Sist. Áudio	Mod.	Freq.	Tipo de Sistema	Adotado
ATSC	DASE[11]	MPEG-2	Dolby AC3[2] (proprietário)	8-VSB[3]	6 MHz	ATSC-C	Estados Unidos, Canadá, Coréia do Sul, Taiwan, México, Argentina
						ATSC-T	
DVB	MHP	MPEG-2	MPEG-2 digital sound (aberto)	COFDM[4]	7 MHz e 8 MHz	DVB-C	Europa, Austrália, Nova Zelândia, Rússia
						DVB-S	
						DVB-T	
ISDB	ARIB	MPEG-2	MPEG-2 AAC	COFDM	6MHz	ISDB-C	Japão
						ISDB-S	
						ISDB-T	

Tabela 1 - Padrões de TV Digital

1.3. OBJETIVOS

Como se demonstrou brevemente nesta introdução, serviços em Televisão Digital Interativa é um assunto muito abrangente e de grande variedade de tópicos. Este trabalho não tem por objetivo especificar todos eles, mas focar-se, principalmente na interatividade. Por conseguinte, faz parte deste uma especificação para desenvolvimento de aplicações, assim como um processo de desenvolvimento do módulo *Gerenciador de Aplicações*.

Explanar-se-á tanto o estado da arte no desenvolvimento de ambos, como inovações a serem agregadas aos mesmos. No campo de inovações serão detalhados possíveis cenários de

2 *Doby AC3*, canais direito, central, esquerdo, *surround* direito e esquerdo independentes + *subwoofer*.
3 *8-Level vestigial sideband modulation*. No canal de 6Mhz, usado no *broadcast*, carrega 19.39 mbps. Usado no codificador Trellis.
4 *Coded Orthogonal Frequency Division Multiplexing*. COFDM pode transmitir vários *streams* de dados simultaneamente, cada uma ocupando uma pequena porçãoda largura de banda total. É o padrão usado na Europa.

aplicação destes serviços, procurando não se constituir como uma mera ficção, mas algo palpável dentro de uma escala de tempo plausível.

1.4. Visão Geral

Procurando ater-se a um fluxo linear de pensamento, em suma, para facilitar o entendimento do explanado, este trabalho conta com uma introdução à luz dos conceitos de Televisão Digital Interativa. Sendo apresentado, em continuidade, o tema abordado.

Os dois principais capítulos são apresentados e descritos em seqüência, primeiramente o desenvolvimento de aplicações para a televisão digital interativa, baseando-se na categorização[15] adotada pelo *DVB-MHP*[08][09]. São apresentadas as descrições requeridas, importantes e desejáveis para se alcançar a correta execução de uma aplicação sobre a plataforma em questão, bem como extensões aos padrões já adotados.

O capítulo seguinte aborda o *Gerenciador de Aplicações*. Neste capítulo é apresentado o conceito de gerenciamento de aplicações, bem como um estudo preliminar de outros gerenciadores. É descrito um processo de elaboração e desenvolvimento do módulo, sendo expostas as necessidades requeridas para se ter um funcionamento mínimo do sistema.

Ao término de cada um destes capítulos é apresentada uma seção de evolução. Nesta são expostos funcionalidades a serem agregadas dentro de um *curto, médio ou longo* prazo.

O Capítulo IV apresenta uma implementação dos módulos críticos de um *Gerenciador de Aplicações*, tendo-se em vista prova dos conceitos apresentados.

Arrematando este trabalho, é apresentada em seu último capítulo uma conclusão, a qual tem seu foco voltado para os possíveis futuros trabalhos.

Este trabalho comporta, ainda, uma rica série de apêndices esclarecendo conceitos e funcionalidades citadas no texto.

Capítulo II

Aplicações

"Algumas das maiores façanhas do mundo foram feitas por pessoas
que não eram suficientemente espertas para saberem que elas eram impossíveis!"

- Doug Larson

As aplicações de TV Digital devem ser portáveis e interpretadas - haja vista a enorme gama de fabricantes de terminais de acesso existentes, utilizando-se de diferentes sistemas operacionais -, bem como de fácil prototipação, já que o cotidiano humano, retratado pelas *Geradoras de Conteúdo*, é substancialmente dinâmico, isto é, a programação de TV pode sofrer uma alteração repentina. Basta que algum fato social tenha capacidade de se transformar em fato histórico, como nos atentados à bomba em Londres neste ano de 2005. Neste caso, as *Geradoras de Conteúdo* deveriam explorar a necessidade atual do telespectador em querer interagir e aprofundar-se na notícia. Desta forma, poderiam ser enviadas aplicações com maiores detalhes dos acontecimentos, com enquetes sobre a crescente onda de terrorismo mundial etc. As possibilidades são inúmeras, visto que estas são limitadas apenas pelo imaginário humano.

Em decorrência das necessidades acima apresentadas, tem-se tornado padrão que as aplicações de TV Digital Interativa sejam desenvolvidas utilizando-se a plataforma Java. Esta tendência, todavia, só se tornou possível no ano de 1999 quando a *Sun Microsystems* divulgou a especificação de JavaTV[14] em sua *Java One Conference*. Até então, Java havia sido introduzida nos *middlewares* pelo padrão aberto MHEG[5] – ainda muito popular no Reino Unido –, contudo não se podia considerar esta implementação como *"pure java"*.

O *DVB-MHP*[08][09] foi o primeiro *middleware* a utilizar Java como base para sua implementação. Este fato torna interessante observar o relacionamento do *MHP* com o padrão *MHEG*, haja vista, que o *MHEG* veio a se tornar o *DAVIC*[6] e que muito das empresas que

[5] *Multimedia and Hypermedia Experts Group* – padrão aberto de TVD Interativa publicado em 1997 pela ISO. Teve muitos padrões tais como MHEG-1, MHEG-3, MHEG-5, MHEG-6 e finalmente o DAVIC.
[6] *Digital Áudio Visual Council* – padrão aberto de TVD Interativa usando como *core* o MHEG-6, o qual nunca foi distribuído.

trabalhavam em sua especificação também trabalharam na especificação do *MHP* – vide Ilustração 7.

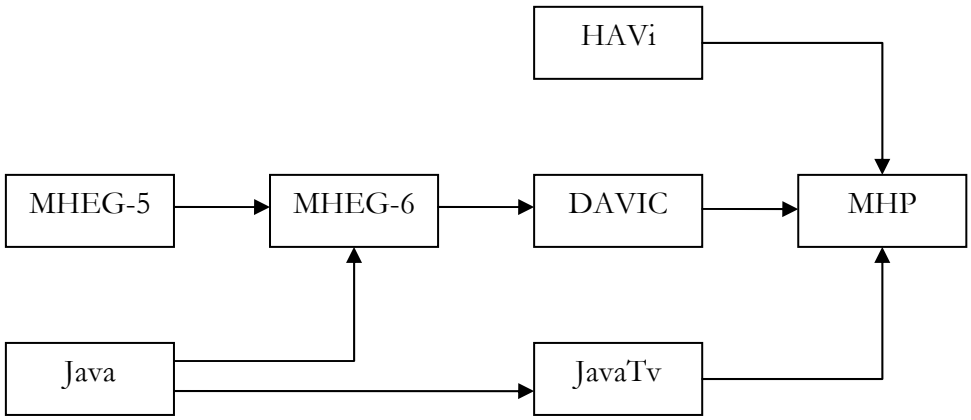

Ilustração 7 - MHP e seus relacionamentos

No *DVB-MHP*[08][09] a definição existente para aplicação é a de que a mesma é uma implementação funcional de um serviço interativo. O *MHP*[09] possui dois níveis de categorização de aplicações. No primeiro as aplicações são separadas quanto ao tipo – procedurais ou declarativas. O segundo nível vem de uma visão de mercado, no qual o consórcio *DVB*[08] definiu três perfis de terminais de acessos, em pretensão de atender as diversas camadas sociais. A saber:

- *Enhanced Broadcast Profile*: visando terminais de aceso mais baratos. Aplicações acessam informações unicamente do *middleware* sobre o qual executam;
- *Interactive Broadcast Profile*: dá suporte ao chamado *Canal de Retorno*. Aplicações podem ser *downloaded* pelo *broadcast* como pelo canal de retorno. As aplicações podem ainda acessar dados remotos através deste canal – que convém ser nomeado de *Canal de Interatividade*;
- *Internet Access Profile*: suporta a aplicações de *Internet*, tais como *e-mail* e navegador *web*.

Existem outras categorizações mais amplas, além destas utilizadas no *MHP*[09], que podem ser aplicadas a todos os *middlewares*:

- Vinculadas a um serviço *(Service-bound)*: nesta categoria encontram-se as aplicações que estão vinculadas a um canal de televisão ou a um evento específico – no DASE pode-se ter aplicações vinculadas a um *range* de canais. Em sua maioria são auto-executáveis. Quase todas estas aplicações chegam ao terminal de acesso através do *broadcast*. São aplicações temporárias, sendo removidas após sua execução.

- Desvinculadas de serviço *(Unbound)*: estas aplicações não mantêm vínculos com serviços ou eventos. Podem chegar ao terminal tanto pelo *broadcast* como através do *Canal de Interatividade*. Podem ser tanto auto-executáveis como necessitarem da requisição para execução vinda do usuário. O usuário pode querer armazenar este tipo de aplicação transformando-a em uma aplicação *storage*.
- Armazenadas no terminal de acesso *(Storage)*: São aplicações *unbound* que foram salvas (armazenadas) pelo usuário. Estas devem aparecer na lista de aplicações existentes no sistema. O usuário pode a qualquer momento removê-las do terminal de acesso.
- Nativas *(Native)*: são aplicações desenvolvidas na linguagem nativa do sistema, padronizadamente C ou C++. Normalmente são produzidas pelo fabricante do terminal de acesso. Estas aplicações não podem ser removidas do terminal, sendo no máximo *updated*.

As aplicações bem como sua categorização representam o foco deste capítulo, baseando as explanações no padrão apresentado pelo *MHP* [09] e pela JavaTV *API* [13].

2.1. Tipos de Serviços Interativos

Como foi introduzido na seção acima, existem várias categorias de aplicações. Esta seção detalha cada uma delas, atendo-se às suas diferenças e apresentando as capacidades funcionais de cada.

Faz-se necessário, entretanto, uma apresentação da forma como estas aplicações chegam aos terminais de acesso, como os mesmos reconhecem as diversas categorias, além de como reconhecer se uma aplicação está apta a executar sobre a plataforma. Para isso deve-se observar que nos padrões atuais, todos os dados descritores das aplicações são enviados em tabelas de informação multiplexadas com o áudio e vídeo do serviço. No caso específico de aplicações, estes dados encontram-se em uma *Application Information Table (AIT)*. Contudo cada padrão especificou os seus próprios dados a serem transmitidos dentro de uma *AIT*, tornando complicado a construção de aplicações portáveis a todos os sistemas de TVDI.

Soluções Propostas

Na busca por uma padronização, apresento duas soluções. A primeira é óbvia: unificação dos dados a serem transportados na *AIT* – vide Tabela 2 –, sendo mais simples de implementar, requerendo pouco retrabalho para os consórcios, porém estando amarrado à uma decisão de mercado. A segunda seria utilizar os dados presentes em cada padrão e portar o restante dentro

de um arquivo descritor do serviço: *descriptor.ds*. Este arquivo estaria compactado junto com a aplicação – sendo esta compactada em um *Jar File* –, seguindo o modelo de pacotes apresentado na Ilustração 8. Informações extras devem vir descritas em um arquivo: *properties.pr*.

Veja que tanto o *Descriptor.ds* como o *Properties.pr* podem ser tanto arquivos de propriedades – apresentados e detalhados na *API* padrão de Java – como arquivos *XML*.

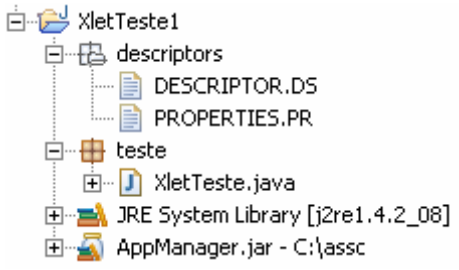

Ilustração 8 - Arquitetura de Pacotes do *Xlet* Proposto

Note que a segunda solução é mais complexa, afetando a estrutura interna dos *Gerenciadores de Aplicações*. Os mesmos terão que analisar os dados da *AIT*, bem como os dados apresentados no arquivo de descrição – a prioridade dada é para os dados contidos neste último. Contudo tal modificação não é tão difícil de ser desenvolvida, já que os *Gerenciadores de Aplicações* atuais já lêem descrições de arquivos de propriedades.

Independentemente de qual solução adotar, seria interessante que as aplicações sejam compactadas. Isso facilita a transmissão das mesmas, certificação de segurança, bem como a interpretação dos dados e gerenciamento das atividades.

Campo	Descrição
application_type	Representa o tipo da aplicação: • Procedural • Declarativa
application_control_ode	Representa o *Status*[7] da aplicação.
protocol_id	Usado para identificação da organização que está provendo os fluxos *IP multicast* usados pela aplicação.
transport_protocol_label	Descreve o protocolo utilizado para carregar a aplicação: • *Object Caroussel* • *IP* encapsualdo em multi-protocolo • *IP* sobre *Canal de Interatividade*
url_locator	Localização dos arquivos.
ISO_639_language_code	Linguagem da aplicação.
application_name	Nome da Aplicação
icon_locator	Localização do arquivo do ícone *(.icon)*
icon_flags	Formato e tamanho do ícone.
base_directory	Nome do diretório no servidor.

[7] O *Status* que cada aplicação pode assumir são: *AUTOSTART, PRESENT, DESTROY, KILL, PREFETCH, REMOTE*.

classpath_extension	*Path* dos arquivos.
xlet_profile	Perfil no qual a aplicação pode ser executada (Segundo nível de categorização.)
application_version	Versão atual da aplicação.
application_size	Tamanho da aplicação.*
application_info	Breve informação sobre a aplicação.*
application_data_size	Tamanho da aplicação em execução.*
application_jar_file	Nome do *Jar File**
application_parameters	Eventuais parâmetros necessários à inicialização da aplicação.*
xlet_initial_class	*MainClass*. Para caso não seja especificado no *manifest.mf* do *Jar File*
declarative_initial_page	O arquivo inicial da aplicação declarativa
declarative_boundary	Descreve o conjunto de páginas declarativas que devem ser consideradas como parte da aplicação, bem como qualquer aplicação interna.
prefetch	Provê dicas para o terminal de acesso sobre quais módulos *DSM-CC*[8] devem ser armazenados na *cache* para melhorar o *start up* das aplicações.
run_background	A aplicação pode continuar em execução mesmo sem estar com o foco. Importante para as novas versões de *Gerenciadores de Aplicações.**
vendor_name	Nome do fabricante. Deve trazer o OID da aplicação.
signed	Se a aplicação é *trusted-signed* por uma Certificadora Digital*
service_bounded	Se a aplicação é vinculada a um serviço – programa.
***Campos propostos**	

Tabela 2 - Dados Presentes em uma AIT

2.1.1. PROCEDURAIS

Aplicações procedurais são aplicações funcionais similares às tradicionais de Java ou C/C++. Contudo possuem certas peculiaridades que as tornam bem diferentes. Uma destas é o seu ciclo de vida diferenciado – lembra o de um *Applet* Java. A JavaTV *API* [13] denomina esta categoria de aplicações como *Xlets*. No *MHP* [09] estas aplicações recebem o nome de *DVB-J Applications*, sendo apenas uma nova nomenclatura, já que na prática são *Xlets* Java.

XLETS

Um *Xlet* é atualmente um conjunto de classes Java que operam em conjunto, devendo ser sinalizadas como uma única instância na base de aplicações do *Gerenciador de Aplicações* [06]. Estas podem entrar em execução por conta própria – no caso de serem sinalizadas como *Autostart* –, ou através da requisição de um *Navegador de Aplicações*.

[8] DSM-CC: Digital Storage Media Command and Control

CICLO DE VIDA DOS XLETS

Existem algumas premissas assumidas e ações a serem efetuadas em cada estado do ciclo de vida de um *Xlet* – vide Ilustração 9. Quando um *Xlet* chega ao terminal de acesso encontra-se no estado *Not_Loaded*, neste ainda não foi carregado na memória da *JVM*[9] nem tem associado a si um *ClassLoader*. Ao assumir o estado *Loaded*, o *Gerenciador de Aplicações* cria uma instância do *Xlet* e associa um Loader ao mesmo. O próximo passo é a inicialização, na qual é passado ao *Xlet* um objeto *XletContext*, que servirá de ponte de comunicação entre o sistema e a aplicação – neste momento a aplicação encontra-se no estado *Paused*. Deste estado, o *Xlet* pode ser requisitado a executar suas funções – passando ao estado *Started* –, no qual competirá por acesso aos recursos compartilhados do sistema. A qualquer momento o *Xlet* pode ser inquirido a terminar sua execução, de maneira condicional ou não.

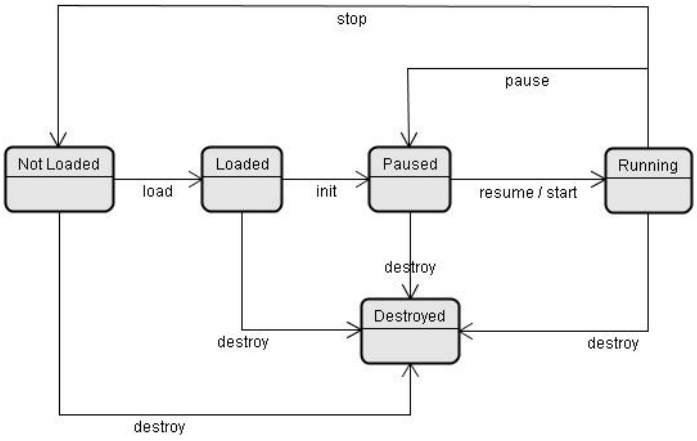

Ilustração 9 – Ciclo de Vida de um Xlet

A cada transição de estado – efetuada com sucesso ou não – é lançado um evento **`AppStateChangeEvent`** para todos os *listeners* cadastrados no *Xlet*. Isto conta ao *listener* qual estado atual do *Xlet* e qual estado anterior.

A implementação de um *Xlet* simples pode ser vista no Apêndice I.

XLETCONTEXT

Quando um *Xlet* é inicializado, recebe um objeto *XletContext*. É através do mesmo que será realizada a comunicação entre o *Gerenciador de Aplicações* e a aplicação. A Ilustração 10 mostra como é feita esta comunicação na prática.

[9] Java Virtual Machine. A máquina que interpreta os *bytecodes* de Java

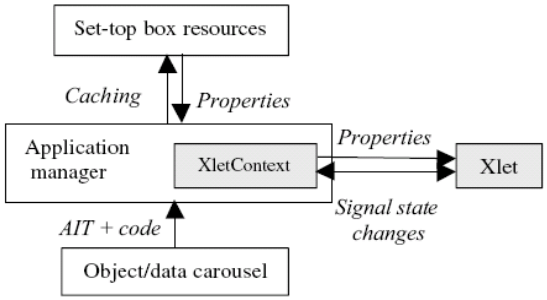

Ilustração 10 - Arquitetura da Comunicação

O objeto *XletContext* serve de interface conversacional bi-direcional. Através dele o *Gerenciador de Aplicações* conhece a aplicação invocadora dos métodos, permitindo ou não o acesso a eles – através de políticas de segurança –, bem como acessa propriedades específicas de cada aplicação. Estas propriedades são carregadas a partir do arquivo *properties.pr*, enviado junto com a aplicação. Alguns dados que devem estar presentes no mesmo são: perfil necessário à correta execução, parâmetros de inicialização, informação adicional, requisição de acesso à área de armazenamento e requisição à manipulação de arquivos *(Leitura/Escrita/Execução)*.

Para as aplicações *Xlets* este objeto provê métodos específicos, tais como:

- *NotifyDestroy*: usado por uma aplicação para notificar ao *Gerenciador de Aplicações* que a mesma entrou no estado *Destroyed*. O *Gerenciador de Aplicações*, então, não deverá invocar o método *destroyXlet* do *Xlet* vinculado a este contexto. O próprio *Xlet* cuidou de que todos os recursos usados pelo mesmo foram liberados, ou seja, o *Xlet* deve ter efetuado a limpeza de suas variáveis, liberação de recursos compartilhados etc.
- *NotifyPaused*: notifica ao *Gerenciador de Aplicações* que o *Xlet* não deve mais permanecer no estado ativo, passando o mesmo para o estado *Paused*. Invocar este método não surtirá efeito algum caso o *Xlet* estiver nos estados *Destroyed*, *NotLoaded* e *Loaded*. Se um *Xlet* passar ao estado *Paused* poderá receber uma chamada para que passe ao estado ativo ou para que seja destruído.
- *ResumeRequest*: permite que o *Xlet* notifique o *Gerenciador de Aplicações* de seu interesse em voltar ao estado ativo, ou seja, voltar a executar. Chamadas a este método podem ser realizadas pelo *Gerenciador de Aplicações* para determinar quais *Xlets* entrarão no estado ativo.

Expansão do XletContext

São muitas as possibilidades apresentadas pelo objeto *XletContext*, todavia o mesmo não supre todas as necessidades. Desta forma, torna-se interessante estender esta interface, provendo novos métodos aos desenvolvedores de aplicações procedurais, tais como:

- *getPlataformData*: Através deste método as aplicações acessam alguns dados referentes a plataforma que estão executando. Por exemplo, quais as aplicações existentes, que tipo de aplicações são – nativas ou *storage*, procedurais ou declarativas –, quais estão em execução, status de cada uma; pode ainda saber qual o tipo de plataforma, se dá suporte ao *Canal de Interatividade* etc.
- *getProxyInStack*: O *Gerenciador de Aplicações* proposto neste trabalho de graduação faz uso de uma pilha de aplicações em execução e/ou pausa temporária. Aplicações podem, através deste método, descobrir quais são as aplicações da pilha.
- *ConversationRequest*: A JavaTV *API*[13] oferece um acesso conversacional com outras aplicações de uma maneira extremamente complicada para o desenvolvedor. Assim, a inserção deste método almeja facilitar o acesso a esta funcionalidade.

Aplicações Nativas

O terminal de acesso pode conter aplicações nativas[10], desenvolvidas em linguagens próximas ao sistema operacional adotado, normalmente C ou C++. Estas aplicações devem possuir o mesmo ciclo de vida apresentado para os *Xlets*. Contudo alguns estados são extremamente complicados de serem alcançados *(Paused)*, ou simplesmente não fazem sentido neste caso *(NotLoaded, Destroyed)*.

Para estas aplicações vale a funcionalidade de interação com outras aplicações, sendo necessário prover interfaces de comunicação entre as mesmas, seja para invocação de métodos como para execução. Desta forma, deve existir, também, um objeto *Contexto* que provenha à interação bi-direcional.

Normalmente estas aplicações constituem o mínimo necessário à utilização de recursos do sistema, podendo pertencer a qualquer tipo de perfil interativo.

[10] Não devem ser confundidas com as chamadas *System Applications*: aplicações manufaturadas pelo fabricante do terminal de acesso. Não são necessariamente apresentadas ao usuário. Aplicações nativas fazem parte de um sub-conjunto destas.

Exemplos de Aplicações Procedurais

Local Interactive

- *Configuração do Terminal de Acesso*: O usuário pode configurar o seu terminal de acesso, personalizando-o com suas preferências: canais e aplicações favoritos e bloqueados, bloqueio de execução de aplicações, perfis de usuário, temas etc. Pode ainda cadastrar dados pessoais a serem utilizados por aplicações de *t-commerce*[11]

- *Navegador de Aplicações*: Aplicação que seve como uma *GUI*[12] para o *Gerenciador de Aplicações*. É através dele que o usuário armazenará, removerá e executará aplicações, bem como receber notificações de erros e mensagens do sistema. Ainda deve provê uma forma de categorização de aplicações para o usuário, tipo: jogos, esportes, escritório etc.

- *Eletronic Program Guide(EPG)*: normalmente é uma aplicação nativa ao terminal de acesso. Pode ser simples, sem uso do *Canal de Interatividade*, ou complexa, utiliza o *Canal de Interatividade* de forma *two-way*. Sua principal função é apresentar um guia completo de todos os serviços, bem como início e término de eventos.

One-Way Interactive

- *Enquetes*: Normalmente *Xlets* que alcançam o terminal de acesso via *broadcast*, sendo vinculados a um serviço ou a um evento específico. Trazem questões simples para ouvir a opinião do usuário e as remetem de volta ao servidor, através do *Canal de Interatividade*.

- *T-commerce*: Aplicações *Xlets* que podem chegar ao terminal de acesso tanto pelo *broadcast* como *donwloaed* através do *Canal de Interatividade*. Podem estar vinculadas ao contexto ou não, podendo, muitas vezes, enquadrar-se em qualquer um dos perfis. O conceito por trás destas aplicações é apelativo, ou seja, puramente comercial.

Two-Way Interactive

- *Browser:* Muitas vezes nativo ao terminal de acesso, provê as mesmas funcionalidades que um *browser* comum.

[11] *Television Commerce*:Novo setor mercadológico criado ao se dispor comércio através da televisão.
[12] *Graphical User Interface*

- *E-mail*: Bem como o *Browser* é normalmente nativo e trabalha de maneira similar ao tradicional programa de gerenciamento de *emails*. Contudo há um grande problema a ser solucionado no caso desta aplicação em específico. O paradigma de *email* preza no *delivery unicast*, o que se for efetuado pelo *broadcast* é impossível. Desta forma cai-se na questão? Ter um gerenciador de *emails* para receber *spans*? Este tópico ainda está sob discussão.

2.1.2. DECLARATIVAS

As aplicações declarativas são versões modernas e reestruturadas dos antigos *"teletextos"*, que se utilizavam do *Vertical Blanking Interval (VBI)* entre as linhas de vídeo para transmitir pacotes de dados e apresentar páginas de texto na tela da televisão[17]. No mundo de TVDI não existe *VBI*. Os dados são transmitidos multiplexados junto com os outros *streams* – como foi explicado no Capítulo I. Além disto, como a largura de banda usado na transmissão do padrão digital de TV é maior, permitindo que este tipo de aplicação possua menus, gráficos coloridos, textos formatados, imagens e até mesmo outras aplicações.

As aplicações declarativas podem ser desenvolvidas em qualquer linguagem de marcadores, tais como HTML, XHTML ou XML, como em linguagens mais complexas como NCL, SMIL, MPEG-4 e MPEG-4 XMT-O. No MHP é utilizado o XHTML como base, denominando suas aplicações como DVB-HTML Applications. O mais interessante é que o suporte a este tipo de aplicação é opcional nos terminais de acesso que tenham o MHP como plataforma de middleware. No Apêndice II apresento como se construir aplicações DVB-HTML.

Dentre as linguagens de marcadores citadas a *XML* é de longe s mais interessante para o desenvolvimento destas aplicações. Os pontos fortes são: fácil integração com Java – existem *APIs* padronizadas –, criação de novas *tags* por parte do desenvolvedor, validação simples – através de *DTDs*[13] e *Schemas*[14] –, bem como padronização e manutenibilidade.

Quando uma aplicação declarativa é sinalizada no terminal de acesso o *Gerenciador de Aplicações* inicializa o processo de *parser* do documento, a fim de criar uma árvore que represente a aplicação. Este processo pode ser visualizado na Ilustração 11 [17].

Como citado anteriormente, é possível que estas aplicações possuam outras internas a si. Estas podem ser tanto declarativas como procedurais. No momento que o usuário selecionar um

[13] Gramática definida pelo desenvolvedor. Explicita relacionamentos, expressões regulares válidas, composições, estrutura das *tags*, formatação do documento, dados válidos. É através dela que é feita a validação do arquivo "*.xml*".
[14] Gramática muito parecida com a *DTD*. Padronizada pela *Microsoft*, tem um poder de definição maior que a *DTD*, contudo por ser um padrão fechado não é largamente utilizado.

link que inicie uma aplicação interna o *Gerenciador de Aplicações* receberá um sinal *(trigger)* para executar a mesma. Importante ressaltar que esta aplicação já deve se encontrar previamente carregada no sistema.

No paper *"A digital teletext service"* Chengyuan Peng e Petri Vuorimaa propõem uma gramática *(DTD)* para a descrição de serviços de *teletexto*. Só que na mesma não há descrição para aplicações internas, sendo necessário uma pequena expansão a fim de suportar esta peculiaridade extra.

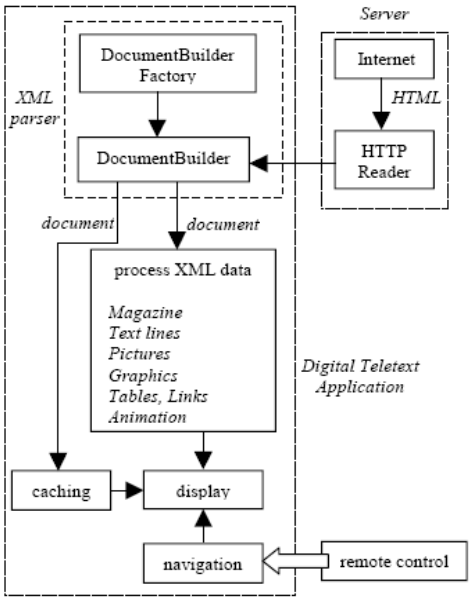

Ilustração 11 - Arquitetura do sistema de Teletexto

Expansão da Gramática

A fim de dar suporte a aplicações internas é necessário expandir a gramática apresentada adicionando um novo elemento. Este elemento é bem semelhante à *tag Object* presente em *HTML*:

<!ELEMENT INNER_APPLICATION			
"Width	*CDATA*	*#REQUIRED*	*-- largura da aplicação --*
Height	*CDATA*	*#REQUIRED*	*-- altura da aplicação --*
CodeType	*CDATA*	*#REQUIRED*	*-- procedural | declarativa --*
ClassID	*CDATA*	*#REQUIRED*	*-- main class | initial page --*
Codebase	*CDATA*	*#REQUIRED*	*-- diretório base --*
JarFile	*CDATA*	*#IMPLIED*	*-- nome do arquivo jar –*
Declare		*#IMPLIED*	*-- define a classe neste instante –*
">			

Tabela 3 - Elemento Inner-Application

```
<!ATTLIST INNER_APPLICATION
        "Param_Name     CDATA       #REQUIRED
        Value           CDATA       #REQUIRED"
>
```

Tabela 4 - Atributo - Lista Inner-Application

É necessário, também, a adição deste novo elemento na entrada do serviço de *teletexto* proposto em [17]:

```
<!ELEMENT DIGITALTELETEXT
                (TITLE, BACKGROUND,
                SERVICE, FONTSTYLE?,
                NEWSFLASH?, COLORBUTTON+,
                MAGAZINE+,  INNER_APPLICATION
                )
>
```

Tabela 5 - Elemento Digital Teletexto – *Expandido*

Exemplos de Aplicações Declarativas

Local Interactive

- *Visualizador de Imagens*: as aplicações declarativas podem ser utilizadas para visualizar imagens de diferentes tipos e formatos. Cada imagem é vista como uma aplicação.
- *E-book*:um livro eletrônico pode ser escrito em uma linguagem de marcadores, sendo, por conseguinte, uma aplicação declarativa. Perceba que o *e-book* pode possuir imagens internas e, que estas, como abordado no tópico acima, são consideradas aplicações, contudo não são consideradas aplicações internas.

One-Way Interactive

- *Cadastros*: o usuário estará sempre se deparando com aplicações que solicitarão o preenchimento de um cadastro – como o exemplo apresentado na introdução, onde o usuário marca um *test-drive*. Estas aplicações declarativas, normalmente internas a aplicações procedurais, necessitam enviar as informações coletadas para o seu servido, via *Canal de Interatividade*.

Two-Way Interactive

- *Quiz*: um programa de perguntas e respostas pode utilizar uma comunicação bi-direcional, onde a cada término de fase os dados são enviados a um servidor. Após processamento das respostas, o usuário poderá estar apto a avançar às próximas fases.

2.2. Um Tipo Especial de Serviço *(Inner Apps)*

O DVB-MHP define um tipo especial de aplicação: Aplicações Internas. Estas são aplicações embarcadas dentro de outras aplicações[21], aumentando o leque de possibilidades na criação das mesmas. Contudo há uma restrição que deve ser respeitada: não se podem ter aplicações internas do mesmo tipo que as aplicações que as contém. Desta forma, uma aplicação procedural só poderá ter aplicações internas declarativas e vice-versa.

Utilizando-se a expansão da gramática demonstrada na seção anterior pode-se adicionar uma aplicação procedural dentro de uma declarativa. Há, ainda, um atributo implícito no elemento *Inner_Application*, o *"declare"*. Este atributo permite que a aplicação seja declarada naquele momento, ou seja, passa a existir, para efeitos de sistema, apenas a partir daquele ponto.

O ciclo de vida destas aplicações é um tanto diferente das outras – principalmente no que se diz respeito à inicialização, à execução e ao término –, já que as mesmas estão vinculadas a uma aplicação externa. Isto não significa que o *Gerenciador de Aplicações* não gerencie seu ciclo de vida, mas que o controle direto destas está sob domínio da aplicação que as contém.

Uma aplicação procedural interna a uma declarativa tem sua inicialização efetuada quando o documento é renderizado, porém sua execução só é efetuada em resposta a um evento *DOM* ou a um *trigger*. Seu término acontece quando ela própria o requisitar ou quando o documento ao qual está atrelado deixar de ser renderizado.

No caso de uma aplicação declarativa interna a uma procedural o ciclo de vida muda novamente. A inicialização e a execução são efetuadas no momento em que é gerado um container que dê suporte a renderização de documentos declarativos. Novamente seu término depende de si própria, ou da resposta a um evento da aplicação que a contém.

2.3. Boas Práticas de Programação *(Xlets)*

Existem algumas boas práticas a serem seguidas pelos programadores de aplicações de TVDI. Algumas estão descritas no **`HelloWorldXlet`** apresentado no Apêndice I, contudo há muitas outras a serem conhecidas. Há sempre uma seção nas especificações dos *middlewares*

apontando uma série do que se deve – e, do que não se deve – fazer durante o desenvolvimento de aplicações. As mais importantes estão compiladas a seguir:

- Aplicações não devem realizar nenhuma ação em seu construtor, absolutamente nenhuma;
- Toda e qualquer inicialização deve ser feita no método *initXlet*. Não carregue grandes estruturas de dados – faça isso apenas no momento da execução –, é necessário lembrar-se que os recursos dos terminais de acesso ainda são escassos;
- Os *middlewares* esperam resposta (comunicação síncrona) na invocação dos métodos do *Xlet*. Portanto, estes métodos devem ser de rápida execução;
- A maioria dos *middlewares* solicita a criação de uma *Thread* dentro do método *startXlet* que execute as suas funcionalidades principais. O *Gerenciador de Aplicações* proposto neste trabalho contorna esta situação, deixando o desenvolvedor livre do transtorno de lidar com criação, execução e término de *Threads*;
- Gerenciamento de recursos é de suma importância no ambiente de TVDI, logo não faça muitas modificações em suas configurações nem os mantenha "presos" por muito tempo. É necessário lembrar-se que os mesmo são compartilhados por diversas aplicações;
- Quando o método *pauseXlet* for invocado a aplicação deve liberar o máximo de recursos compartilhados que esteja usando, limpar suas variáveis e esconder qualquer interface gráfica – utilize o método *dispose()*;
- Não se utilize de métodos de sistema, tais como: *System.exit()*,*System.out.*, System.err.**.
- Caso tenha criados outros *Threads* durante sua execução, destrua todos no momento em que o método *destroyXlet* , além de descadastrar-se dos eventos o qual era um *listener*.
- Não tente "pegar" o lançamento da exceção **`ThreadDeath`**. Ela é para uso apenas do *Gerenciador de Aplicações*. Na realidade, não tente pegar nenhuma exceção de **`RunTime`**.

2.4. Apresentando Uma Aplicação Na Tela

Quase todas as aplicações possuirão interfaces gráficas através das quais permitirão a interação com o usuário. No mundo de TVDI, estas interfaces serão desenvolvidas utilizando-se uma *API* que estende o *Abstract Windowing Toolkit (AWT)* de Java, denominada de *HAVi UI API (Home Audio and Video Interoperability User Interaction)*

O *HAVi*[22] define novas classes Java a fim de implementar toda a interface gráfica necessária para TV digital. Divide a arquitetura das interfaces em três planos: *background*, vídeo e *graphics* – vide Ilustração 12. Cada um possui suas próprias configurações, sendo que para o desenvolvedor de aplicações, apenas o *layer*[15] gráfico é realmente importante, já que é no mesmo que adicionará seus componentes.

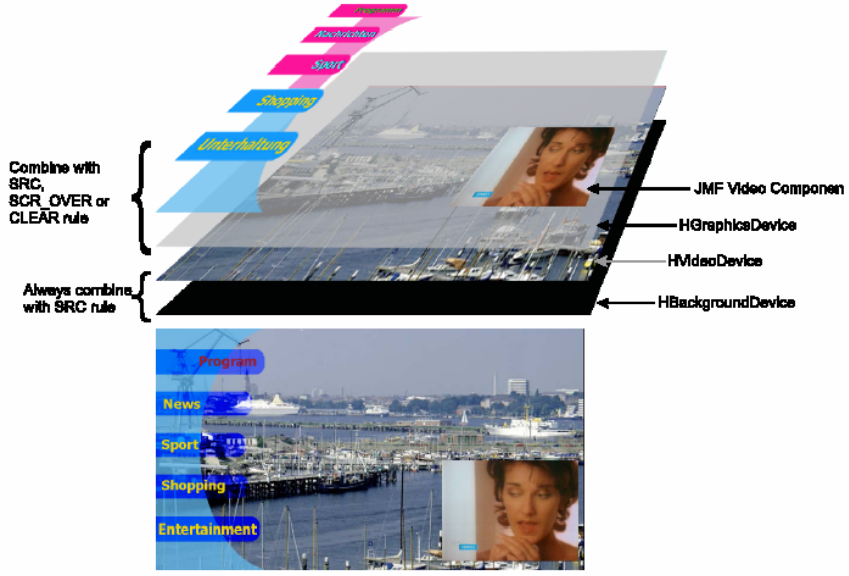

Ilustração 12 - Planos do Havi

Para tanto a aplicação solicitará uma instância da classe **`HScene`**, através de uma **`HSceneFactory`**. Isto capacita o *Xlet* a adicionar e remover seus componentes no plano gráfico.

A *API* de *HAVi* também fornece maneiras de se cadastrar como *listeners* de eventos gráficos, ou seja, *cliques* em botões, ganho e perca de foco etc.

2.5. Evolução

A evolução das aplicações encontra-se vinculada à evolução dos terminais de acesso e de seus componentes – principalmente o *middleware* e o seu módulo de *Gerenciamento de Aplicações* –, já

[15] Camada

que o processo de desenvolvimento adotado é o mesmo para uma aplicação tradicional. Em uma visão de futuro, planejando para curto, médio e longo prazo, vislumbra-se diversas perspectivas de aplicações.

CURTO PRAZO

Numa perspectiva otimista é possível que dentro de pouco tempo os terminais de acesso tornem-se mais robustos, aumentando sua capacidade de armazenamento e de processamento. Isto permitirá que tanto aplicações residentes como aplicações *downloaded* sejam um tanto mais complexas.

Neste cenário poderiam surgir novas versões do *Eletronic Program Guide (EPG)* – com recursos como *PVR*[16], aprendizado de ações, *PIP* e gerenciamento de informações –, do *Visualizador de Imagens*, do *Navegador de Aplicações* etc. Também seriam introduzidas outras aplicações nativas, que permitissem um melhor gerenciamento dos aparelhos domésticos da casa – a chamada casa inteligente, onde o terminal de acesso faria o papel de gerenciador.

MÉDIO PRAZO

Olhando para um médio prazo poderíamos ter *providers* de aplicações acessíveis por meio dos terminais de acesso. Seria necessário padronizar estes serviços, bem como unidades certificadoras para dar confiabilidade e segurança aos mesmos.

Num ambiente como este, poder-se-ia disponibilizar um recurso de personalização total dos terminais de acesso. Essa aplicação contaria com servidores, em cluster, na rede. Estes armazenariam as preferências de cada usuário do sistema, o que permitiria um *login* único em todos os terminais de acesso, com recuperação de toda a personalização pertencente àquele usuário.

Este tipo de aplicação está em estudo pelo consórcio brasileiro de televisão digital *SBTVD*, especificamente pela RFP-04, sendo, atualmente, denominada de *Gerenciador de Perfis de Usuário*.

[16] *Program Video Recorder* – Função de gravação do *stream* selecionado.

LONGO PRAZO

Olhando para um longo prazo poderíamos ter redes de interação formadas por terminais de acesso. Num ambiente com este poderíamos ter aplicações com compartilhamento de recursos, serviços de troca de arquivos, aplicações distribuídas e, até mesmo, com processamento distribuído.

As aplicações poderiam ser descritas em diversas linguagens, sendo transmitidas apenas suas funcionalidades, seus métodos, relacionamentos, e interfaces. Esta descrição, provavelmente desenvolvidas em arquivos *XML*, seria criada através de um codificador – residente na emissora – e, interpretada por um decodificador – residente no terminal de acesso –, que geraria as aplicações na linguagem nativa da máquina.

Capítulo III

Gerenciador de Aplicações

"Disciplina sem liberdade é tirania,
liberdade sem disciplina é o caos."
- Cullen Hightower

O *Gerenciador de Aplicações* é por definição uma parte do sistema de software e residente dentro do terminal de aceso [06]. Sendo responsável pela avaliação do código e da integridade dos dados, sincronização de comandos e informações, adaptação e representação do formato gráfico apropriado para a plataforma de display, obtenção e descarte de recursos de sistema, gerenciamento de erros e sinalização de exceções, inicialização e término de qualquer nova sessão, além do gerenciamento de acesso às variáveis e aos conteúdos compartilhados do sistema[03].

É função, ainda, deste módulo do *middleware*: o controle do ciclo de vida de aplicações, através do gerenciamento de seus estados; a atribuição de prioridades às mesmas, bem como identificação de eventuais falhas e ações para contigenciá-las; e, a definição de um protocolo de comunicação entre as aplicações e o sistema.

Este módulo deve ser posto no *start-up* da máquina, constituindo sempre um processo do sistema – daemon – de baixa prioridade.

3.1. Estado da Arte

Cada *middleware* existente hoje no mercado possui seu próprio *Gerenciador de Aplicações*, bem como especificações de quais aplicações podem executar sobre sua plataforma. Dois dos mais avançados consórcios nesta área são o OCAP e o MHP.

3.1.1. OCAP

O OCAP produziu seu gerenciador junto com o MHP, portanto suas funcionalidades são muito similares. Contudo há sempre algumas diferenças. Por exemplo, os estados assumidos

pelas aplicações durante o ciclo de vida têm nomes diferentes com: *Unloaded, Loading, Running* e *Suspended.*

No OCAP existe uma aplicação *downloaded* denominada de *Monitor Applications* a qual faz uma ponte entre o *network operator* e o terminal de acesso. O *network operator* é quem comanda as ações do terminal de acesso do usuário.

O OCAP instituiu o conceito de aplicação compartilhada – as quais podem estar vinculadas a mais de um serviço, no caso a mais de um canal de televisão –, além de dar suporte a aplicações não vinculadas a um serviço específico, bem como a aplicações descritas em outras linguagens, contanto que as mesmas sejam nativas ao terminal de acesso. Suporta, também, as denominadas aplicações *stored* contanto que as mesmas sejam desvinculadas de serviço.

As aplicações alcançam o terminal de acesso ou via broadcast, ou via conexão *IP*, sendo categorizadas várias aplicações para este último meio de transporte. O suporte a aplicações declarativas é fraco, sendo opcional a sua disponibilidade no terminal de aceso.

Constitui, com algumas modificações, o núcleo do *middleware* DASE do padrão ATSC.

3.1.2. MHP

O MHP em sua primeira versão *(realease 1.0)* era bastante limitado, sem dar suporte a aplicações declarativas. Entretanto em sua nova versão inseriu o conceito de *plugins*, e especificou as chamadas *DVB-HTML Applications*, permitindo não só a execução de aplicações declarativas implementadas nos padrões tradicionais de *web*, como àquelas escritas em linguagens mais complexas. Infelizmente, esta especificação não é obrigatória de implementação.

É totalmente desenvolvido em Java, suportando apenas aplicações desenvolvidas nesta plataforma, desde que as mesmas sejam sempre vinculadas a um, e somente um, serviço. Desconhece por completo a categoria de aplicações desvinculadas de serviço.

O usuário do MHP possui um maior controle de seu terminal, não havendo o *"Monitor Application"* do OCAP. As aplicações podem alcançar o terminal de acesso tanto via *broadcast* como por meio de conexões *IP*. Uma aplicação pode ser armazenada pelo usuário - *stored.*

3.1.3. Comparação Enrtre OCAP e MHP

	OCAP	MHP 1.1
Aplicações Java	Sim	Sim
Aplicações HTML	Sim	Sim
Aplicações declarativas complexas	Não	Sim
Download via Broadcast	Sim	Sim
Download via IP	Sim	Sim
Aplicações Vinculadas a Serviço	Sim	Sim
Aplicações Desvinculadas a Serviço	Sim	Não
Stored pelo Usuário	Não	Sim
Monitor de Aplicações	Sim	Não
Aplicações Nativas	Sim	Não
Plugins	Não	Sim
Aplicações Compartilhadas	Sim	Não

Tabela 6 - Comparação entre OCAP e MHP

3.2. Gerenciador Proposto

Durante o estudo sobre outros *Gerenciadores de Aplicações* tornaram-se evidentes as limitações existentes em cada um, quando isolados. Contudo, ao se unir os conceitos e funcionalidades apresentados neles, têm-se um *Gerenciador de Aplicações* mais abrangente, com um maior poder de execução. Isto aumenta o grau de extensibilidade do sistema, além de facilitar a manutenibilidade e adaptabilidade do mesmo.

Visando alcançar as características supracitadas, bem como introduzir algumas inovações para capacitar os *Gerenciadores de Aplicações* a suportar a nova safra de aplicações – cada vez maiores e mais complexas, inseridas em um cenário diferenciado e de vanguarda – é que me proponho a especificar um *Gerenciador de Aplicações*. Este integrará aplicações internas, sistema de segurança e certificação, adaptação via *plugins*, atuará como servidor de serviços, tendo capacidade de integração em rede e uma pseudo-inteligência.

Para um maior detalhamento dos casos de uso e requisitos do sistema, faz-se necessário o uso do Apêndice III.

3.2.1. Casos de Uso

Um *Gerenciador de Aplicações* tem de prover obrigatoriamente serviços de controle do ciclo de vida de aplicações, bem como meios de recepção de notificação de chegada das mesmas. Além destes recursos básicos, o sistema pode – e deve – prover outros serviços, como: configuração de seu modo de execução, obtenção de dados inerentes ao terminal de acesso, bem como acesso às variáveis e recursos do sistema.

Estes serviços constituem os casos de uso do sistema e subsequentemente seus requisitos funcionais. Contudo, a disponibilidade destes serviços compete aos cenários de utilização, ou seja, a quem necessita invocar o uso dos mesmos, denominados de *Atores do Sistema*.

Atores

Os atores do sistema são as aplicações que executam sobre a plataforma, o próprio *Gerenciador de Aplicações*, ao realizar chamadas recursivas – dependendo do estado sistema –, o *Módulo de Processamento de Dados* do *middleware* – responsável pela interpretação dos dados advindos do broadcast e montagem das aplicações –, e o usuário da TV Digital Interativa. Todos se encontram listados na Tabela 7.

Ator/Usuário	Descrição
Aplicações	Usará o *Gerenciador de Aplicações* para controlar o seu ciclo de vida, bem como para obter informações relevantes do sistema, além de permissões de acesso e de execução.
Gerenciador de Aplicações	O *Gerenciador de Aplicações* pode fazer uso de suas próprias funcionalidades dependendo do estado sistema.
Módulo de Processamento de Dados	Usará o *Gerenciador de Aplicações* para carregar uma aplicação recém instalada no sistema.
Usuário TVDI	Recebe informações do sistema, além de salvar, por em execução, navegar e remover aplicações.

Tabela 7 - Atores do Sistema

Cenários

Os cenários representam as circunstâncias em que cada serviço provido pelo sistema será invocado e, por quem será feita esta requisição. Cada cenário é modelado a partir do *Ator* a interagir com os serviços.

Aplicações:

Uma aplicação, uma vez presente no terminal de acesso e em estado de execução, pode acessar alguns serviços providos pelo *Gerenciador de Aplicações*. Esses serviços, apresentados na Ilustração 13, só se encontram disponíveis a aplicações *auto-executáveis*, certificadas e confiáveis, ou seja, precisam ter sido validadas pelo *Gerenciador de Aplicações*.

Esta validação é requerida em virtude de se evitar que uma aplicação daninha (sem certificado) realize operações ilegais no sistema, sendo sempre efetuada antes que a mesma inicie sua execução.

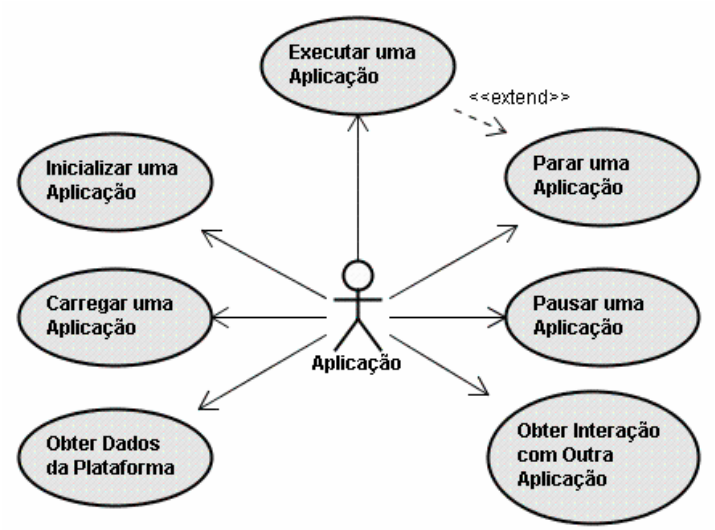

Ilustração 13 - Cenário da Aplicação

***Gerenciador de Aplicações*:**

O próprio sistema pode requerer o uso de serviços referentes ao gerenciamento do ciclo de vida das aplicações. Isto acontecerá apenas se o *Módulo Inteligente* estiver ativado, possibilitando ao sistema analisar se é possível gerenciar duas ou mais aplicações ao mesmo tempo, bem como manusear uma pilha de aplicações que executam em *background*.

Além destes serviços, o *Gerenciador de Aplicações* pode ser obrigado a acessar a rede em busca do *provider* que disponibiliza a aplicação. Seja por que a mesma foi sinalizada na *AIT* como remota, seja por que sua versão não condiz com o terminal de acesso e na *AIT* está sinalizada que existem outras versões a serem *downloaded*.

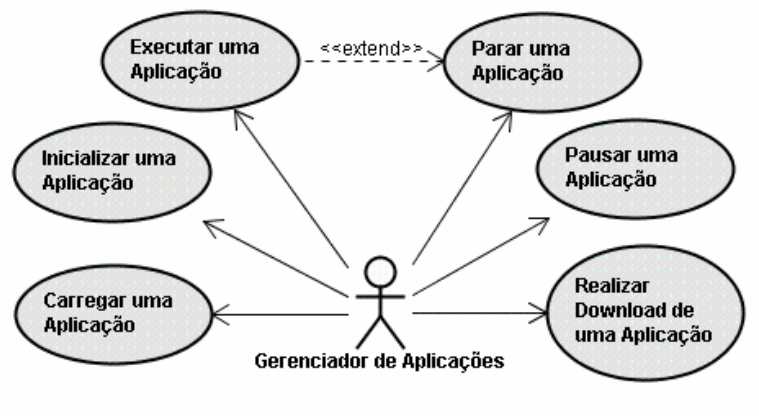

Ilustração 14 - Cenário do Gerenciador de Aplicações

<u>Módulo de Processamento de Dados</u>:

Em todos os *middlewares* estudados encontra-se um módulo que recebe apenas o fluxo dados do *broadcast* e interpreta se há, ou não, aplicações multiplexadas no sinal. Este módulo é responsável pela montagem da aplicação, já que a mesma é transmitida em partes, sendo enviadas através de um mecanismo denominado de *Caroussel*. Cada uma destas partes recebe o nome de *Object Caroussel*.

Quando o *download* de uma aplicação é finalizado, o *Gerenciador de Aplicações* é notificado. Para que este processo seja realizado, o mesmo deve ter se cadastrado com *listener* na interface provida pelo *Módulo de Processamento de Dados*.

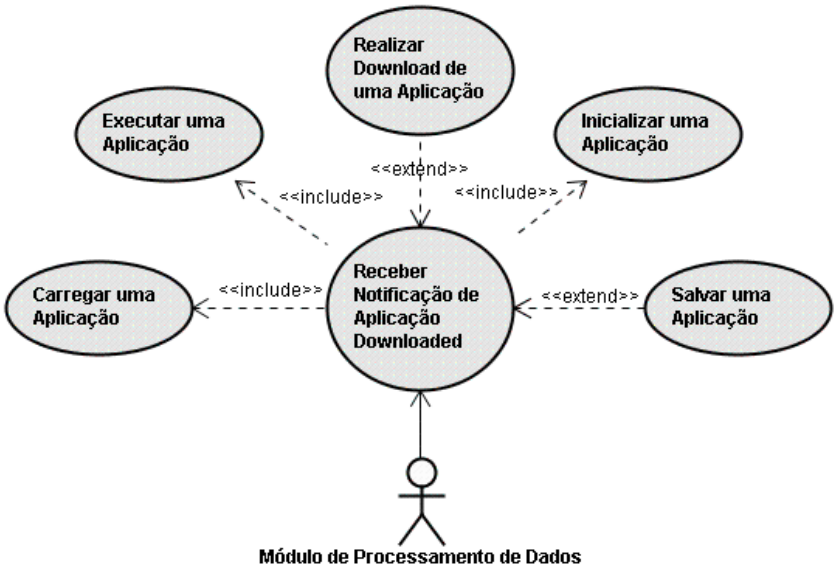

Ilustração 15 - Cenário de Processamento de Dados

No momento em que for notificado da chegada de uma nova aplicação o *Gerenciador de Aplicações* verifica as preferências do usuário e executa a correta seqüência de ações. Normalmente

a aplicação é colocada em execução, ou o usuário recebe uma notificação de que há uma nova aplicação a ser executada.

Usuário TVDI:

O usuário, através das aplicações internas providas pelo *Gerenciador de Aplicações* pode inquirir várias requisições ao sistema, como: limpar toda a tela – parando a execução de toda e qualquer aplicação. Uma das mais importantes, entretanto, é a configuração do módulo. É a partir desta que o *Gerenciador de Aplicações* toma várias decisões e pratica a que mais convém.

Outro importante serviço prestado ao usuário é o de poder recuperar uma aplicação. Este serviço permite que uma aplicação pausada pelo *Gerenciador de Aplicações* possa ser recuperada e colocada de volta à execução. Perceba que isto só é possível porque o sistema guarda estas aplicações em uma pilha. É permitido que o usuário escolha, a partir de uma lista de ícones apresentada na tela, qual aplicação deseja voltar a executar.

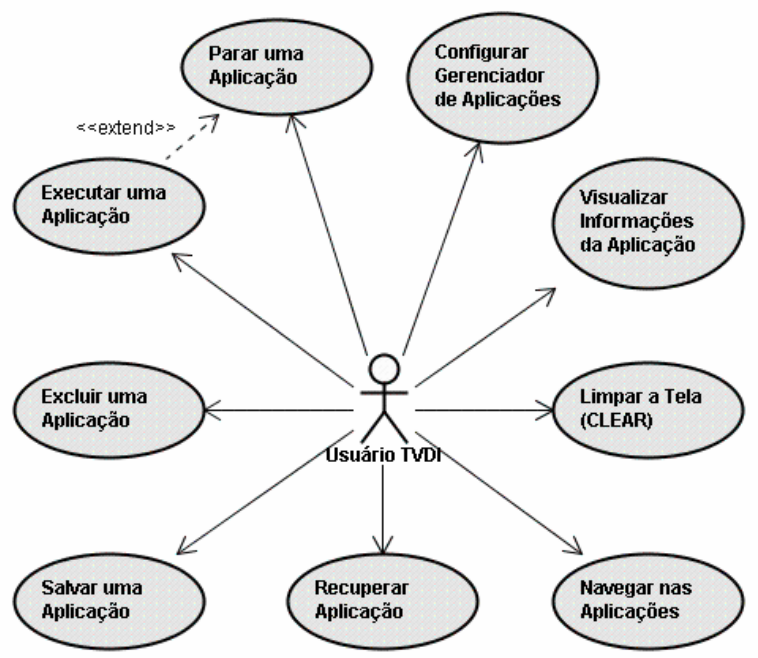

Ilustração 16 - Cenário do Usuário TVDI

COMPONENTES

Um *Gerenciador de Aplicações* deve ser desenvolvido em componentes visando cumprir os requisitos funcionais de manutenibilidade, extensibilidade e adaptabilidade. Desta forma, o sistema aqui proposto, é assim modelado. Note-se que, as aplicações de TVDI estão sendo

desenvolvidas em Java, tornando imprescindível que este módulo do *middleware* acompanhe esta tendência e, seja desenvolvido na mesma linguagem.

Cada um dos componentes apresentados na Ilustração 17 serão detalhados e explicados.

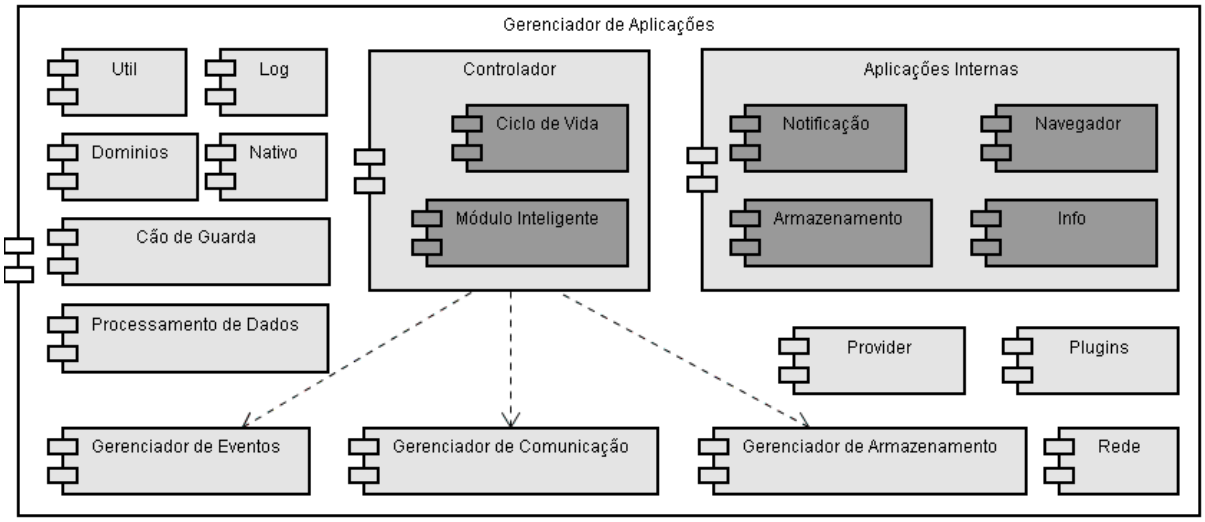

Ilustração 17 - Componentes do Sistema

<u>Aplicações Internas</u>:

Este componente comporta todas as aplicações internas ao *Gerenciador de Aplicações*. Cada aplicação por si só também é um componente, o que permite a substituição de todo o módulo ou de uma aplicação específica.

As aplicações deste componente devem ser desenvolvidas na mesma linguagem do *Gerenciador de Aplicações*, ou seja, Java. Isto facilita a integração e manutenção do sistema, bem como a eventual substituição de componentes. Todavia nada impede que outras linguagens – como C ou C++ – sejam empregadas. Havendo, entretanto, um trabalho extra no uso de *Java Native Interface (JNI)* para integrá-las ao sistema. Desta forma, preza-se que as aplicações internas sejam *Xlets*, tendo seu ciclo de vida controlado pelo próprio *Gerenciador de Aplicações*.

Há uma grande variedade de aplicações internas possíveis, apresento aqui as mais importantes. Saliento que a aplicação, usualmente nativa, *Navegador de Aplicações* foi inserida neste contexto, por resumir-se a uma interface gráfica do *Gerenciador de Aplicações* apresentada ao usuário.

- *Armazenamento*: deve ser executada a fim de questionar o usuário se deseja *"salvar"* uma determinada aplicação. A mesma deve dispor ao usuário a possibilidade de incluí-la em uma categoria de aplicações específica. Pode-se ainda configurar uma categoria geral, onde serão efetuados todos os armazenamentos.

O *Controlador* será notificado assim que uma aplicação for armazenada, delegando ao componente interno *Processamento de Dados* o serviço de acrescentar a aplicação no arquivo de aplicações existentes. É necessária, também, a atualização da *AIT* interna controlada pelo *Gerenciador de Aplicações*.

- *Info*: apresenta mensagens do *Gerenciador de Aplicações* na tela. Podem ser mensagens de erro, informações gerais ou questionamentos de *"sim"* ou *"não"*.

- *Navegador de Aplicações:* Este aplicativo interno dará ao usuário controle sobre as aplicações existentes no sistema, podendo instalar, desinstalar, executar, navegar e obter informações das mesmas. É possível, ainda, inserir a aplicação em uma categoria, bem como gerenciar o processo de cadastro destas categorias.

  Esta aplicação provê uma interface de configuração do sistema. É através da mesma que o usuário poderá escolher entre visualizar ou não as informações sobre as aplicações, visualizar as aplicações em lista ou ícones, ativar ou não o Módulo Inteligente do *Gerenciador de Aplicações*, dentre outros.

- *Notificação*: notifica ao usuário – essa notificação é normalmente feita através de um ícone apresentado no canto direito superior da tela – da chegada de uma nova aplicação. O usuário pode escolher visualizar a descrição da mesma, executá-la ou simplesmente descartá-la.

Cão de Guarda:

Este componente pode ser tanto um dispositivo de hardware – recomendado – como um *software*. No primeiro caso basta adicioná-lo ao sistema e configurá-lo. Sendo o segundo caso um pouco mais complexo.

O *Cão de Guarda* ou *Watchdog Timer* faz parte dos requisitos não-funcionais de robustez, estando descrito no Apêndice III.

Controlador:

Este é o núcleo do *Gerenciador de Aplicações*. As principais funções estão aqui contidas. É a partir deste componente que as decisões são tomadas e as ações delegadas aos respectivos componentes.

O controlador é responsável por notificar o *Cão de Guarda* da correta execução do sistema, por gerenciar o ciclo de vida das aplicações, inicializar todos os componentes, carregar

arquivos e variáveis, associar domínios de segurança específicos a cada aplicação, disponibilizar acesso aos recursos e variáveis compartilhadas do sistema e delegar ações.

Todo *Gerenciador de Aplicações* deve possuir sua própria *AIT* e a mesma deve ser mantida e gerenciada pelo *Controlador*. Logo em sua inicialização este componente deve carregar a *AIT* a partir de um arquivo de configuração – aconselha-se o uso de *XML* –, usando o componente interno de *Processamento de Dados*.

Para cada aplicação sinalizada na *AIT* o *Controlador* associa um *proxy* e adiciona *listeners* a eventuais mudanças de estado deste *proxy*. Um destes *listeners* é o módulo do *middleware* que cuida dos eventos de usuário. Além dos *proxys* a *AIT* manuseada pelo *Controlador* deve possuir registros das aplicações que estão em execução, quais os tipos e seus estados.

Este componente possui dois outros internos, a saber:

- *Ciclo de Vida*: este componente cuida do ciclo de vida das aplicações, sejam procedurais ou declarativas, nativas ou não, vinculadas a serviço ou não e, ainda auto-executáveis ou não. Segue o modelo apresentado pela JavaTV API [13], apresentando extensões, a fim de comportar todas os tipos de aplicações, inclusive as declarativas mais complexas – vide Ilustração 18.

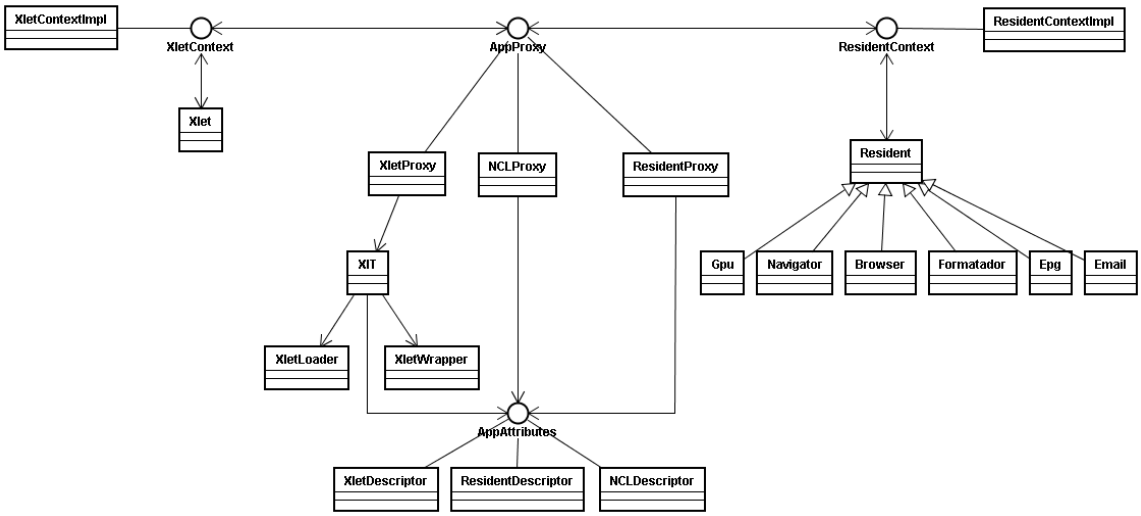

Ilustração 18 - Diagrama de Classes do Ciclo de Vida - FlexTV

As aplicações residentes aqui mostradas estão presentes na proposta de *middleware* apresentada pelo consórcio *FlexTV* ao projeto *SBTVD – Sistema Brasileiro de Televisão Digital*.

Outro ponto importante a ser discutido é a modificação da forma de execução das aplicações *Xlets*. Como explicado na seção *Boas Práticas de Programação* do capítulo

Aplicações, a maioria dos *middlewares* solicita ao desenvolvedor de aplicações que em seu método *startXlet* crie uma nova *Thread*. Esta deve comportar a real execução da aplicação. Isso se faz necessário por que este método é síncrono e o sistema precisa retomar o controle de operação. Todavia, isto é extremamente perigoso, podendo levar o sistema a um *deadlock*.

A solução adotada no projeto *FlexTV* é a criação, pelo *Controlador,* de uma *Thread* própria que encapsule o *Xlet*: o **`XletThread`**. Isto livra o desenvolvedor da aplicação de lidar com *Threads* e possibilita um maior controle por parte do *Gerenciador de Aplicações*. Essa característica permite, também, a criação de domínios de segurança específicos para cada tipo de aplicação.

- *Módulo Inteligente*: este componente constitui a pseudo-inteligência do *Gerenciador de Aplicações*. Este módulo é ativado através do ajuste de uma preferência provida ao usuário pela aplicação interna *Navegador de Aplicações*.

  Com este módulo ativado, o *Gerenciador de Aplicações* manuseia uma pilha de aplicações; provê recuperação das aplicações existentes nesta pilha; executa aplicações em *background*; acessa a rede à procura de aplicações similares, desde que estas tenham versões suportadas pelo terminal de acesso; aprende quais aplicações são as favoritas do usuário; ativa o componente *provider*.

Domínios:

Visando a segurança do sistema – impedindo que aplicações não certificadas como *trusted signed* na *AIT* não danifiquem ou acessem dados importantes do sistema – é necessário criar-se domínios de permissão.

Primeiramente deve-se criar um arquivo *"java.policy"* e carregá-lo no *start up* do sistema. Esse arquivo deve permitir acesso a tudo que for possível de se acessar no sistema. O segundo passo é criar extensões da classe **`SecurityManager`** do pacote java.lang. Cada extensão representa um domínio de permissões, os quais podem ser associados a uma aplicação no momento em que à mesma entre em execução. É preciso utilizar-se deste artifício por que Java só permite um **`SecurityManager`** por vez.

O desenvolvedor de um *Gerenciador de Aplicações* pode construir vários domínios de permissão, contudo alguns são estritamente necessários:

- *Aplicações Residentes e Internas*: este domínio utiliza o chamado **`NullSecurityManager`**. Esta classe libera acesso a tudo que estiver liberado pelo arquivo *"java.policy"*.

- *Aplicações Trusted-Signed Autoexecutáveis*: deve-se estender a classe **`NullSecurityManager`** impedindo acesso a métodos internos ao *middleware*, mas capacitando acesso a todos os métodos das interfaces **`XletContext`** e **`AppProxy`**. Isto permitirá que uma aplicação possa solicitar dados da plataforma, e executar outras aplicações, bem como acessar *APIs* que lhe dêem acesso a rede.

- *Aplicações Trusted-Signed*: este domínio comporta aplicações certificadas, mas que não são auto-executáveis. O que pelo padrão de JavaTV não lhes dá permissão de executar outras aplicações, nem acessar os métodos estendidos na classe **`XletContext`**. A classe que herdará de **`NullSecurityManager`** deve prover este tipo de policiamento.

- *Aplicações Sem Cerificado*: este é o domínio mais pobre. A aplicação mesmo que precise realizar acesso à rede não poderá, estando restrita a interatividade local. Este domínio não deve permitir acesso a arquivos, no máximo leitura; deve impedir acesso a recursos escassos etc.

Gerenciador de Armazenamento:

Nos *middlewares* sempre há uma área de armazenamento temporária, podendo, também, haver uma de caráter permanente aberto ao acesso das aplicações. Este módulo gerencia o acesso a estes recursos, criando *pools* de conexões, e lançando eventos associados a requisições e liberações dos mesmos.

Este componente também trata do armazenamento das aplicações, sua indexação, além de métodos de busca, inserção, remoção e atualização.

Gerenciador de Comunicação:

É uma interface entre o *Gerenciador de Aplicações* e o módulo do *middleware* que gerencia a comunicação inter-aplicações. Na realidade, o que este componente faz, é repassar as referências das aplicações que solicitam interação para o módulo responsável, ficando livre, o *Gerenciador de Aplicações*, de gerenciar a troca de mensagens entre as mesmas. Contudo o ciclo de vida destas aplicações ainda continua sobre a tutela do *Gerenciador*.

Gerenciador de Eventos:

Este componente é um *listener* de eventos cadastrado por código no módulo responsável pela captura de todos os eventos de usuário. Contudo o mesmo não trata todos os eventos[17], ficando responsável apenas por alguns deles. Exemplo:

- *VK_CLEAR*: Notifica o *Controlador* do desejo do usuário em limpar todas as aplicações em execução, estejam visíveis ou não e, acessar apenas o *streams* de áudio e vídeo*;*
- *VK_GUIDE*: Requisita ao *Controlador* a colocar a aplicação residente *Eletronic Program Guide* em execução, como guia de programação*;*
- *VK_INFO*: Requisita ao *Controlador* a colocar a aplicação residente *Eletronic Program Guide* em execução, como base de informação sobre o serviço*;*
- *VK_MENU*: Requisita ao *Controlador* a carregar o *menu* do terminal de acesso na tela do usuário. Este *menu* pode ser tanto uma aplicação nativa como fazer parte da aplicação interna *Navegador de Aplicações*.
- *VK_NAVIGATOR*: Requisita ao *Controlador* a execução da aplicação interna *Navegador de Aplicações*.
- *VK_TELETEX*: Requisita ao *Controlador* a execução da aplicação residente de *Teletexto* responsável pela exibição de aplicações declarativas

É factível que este componente possa tratar outros eventos, como por exemplo: a junção de teclas a fim de acessar a recuperação de aplicações, caso o módulo inteligente esteja ativado, ou outras teclas do controle remoto. Fica livre a implementação do tratamento de eventos.

Log:

Componente responsável por armazenar todo o *log* do sistema. Este *log* pode ser acessado pelos fabricantes para manutenção do equipamento, verificando possíveis erros. Há uma opção a ser disponibilizada no *Navegador de Aplicações*: participar do sistema de aprimoramento do equipamento pelo fabricante. O *log* será enviado a cada intervalo tempo pré-definido pelo fabricante.

[17] Os eventos aqui apresentados são eventos HAVi

Este *log* é gerenciado pelo *Controlador* sendo removido uma parte do mesmo em intervalos regulares de tempo. Esta é uma prática importante a ser realizada, sempre se tendo em mente que o espaço de armazenamento do sistema é escasso.

Para implementar este *log* pode-se utilizar a API de *log* provida por Java. Nesta, pode-se ajustar várias características, dentre elas: nível do *log* e o formato de armazenamento.

Nativo:

Este componente é responsável por toda e qualquer interface nativa entre o *Gerenciador de Aplicações,* o *middleware* e as aplicações nativas. Totalmente desenvolvido na tecnologia *Java Native Interface (JNI)*, portando capacidade de conversação bi-direcional ao sistema.

É importante ressaltar que o uso de *JNI* na conversação entre dois softwares, sendo um Java e um outro C, pode ser feito de duas formas:

- *Java inicia a conversação (Recomendado)*: nesta primeira maneira, um processo de Java inicia a comunicação com um processo de C, passando para o mesmo, como parâmetro, o ambiente no qual executa, ou seja, um ponteiro para a máquina virtual na qual está inserido. Esta forma é a mais recomendada, já que isto seria feito no momento de se inicializar as aplicações residentes, compactuando com o ciclo de vida esperado para as mesmas.
- *C inicia a conversação*: o processo C cria um processo para a máquina virtual Java e solicita a execução do aplicativo Java. Esta forma deve ser descartada, já que não convém possuir vários processos de máquinas virtuais Java num mesmo terminal de acesso.

***Plug-ins*:**

O MHP trabalha o conceito de *plugins*, o qual também foi inserido nesta proposta. *Plug-ins* são formas de portar aplicações – descritas para executar em outras plataformas – para a plataforma desejada.

Desta forma, ao se utilizar deste conceito, o *Gerenciador de Aplicações* aqui proposto poderia executar aplicações descritas para qualquer outro *middleware* contanto que o desenvolvedor criasse um *plug-in* que portasse sua aplicação para este sistema.

O MHP classifica os *plug-ins* em interoperáveis e não-interoperáveis. Contudo não é dado suporte ao segundo grupo, o qual deve ser definido pelo desenvolvedor do *middleware*. O primeiro

deve ser escrito em Java e é amplamente especificado no MHP 1.1, devendo o desenvolvedor analisar a fundo esta especificação.

É importante ressaltar que o ciclo de vida de aplicações que precisam de *plug-ins* é diferenciado. Por conseguinte o MHP introduziu três novos estados referentes aos *plug-ins*, como mostra a Ilustração 19.

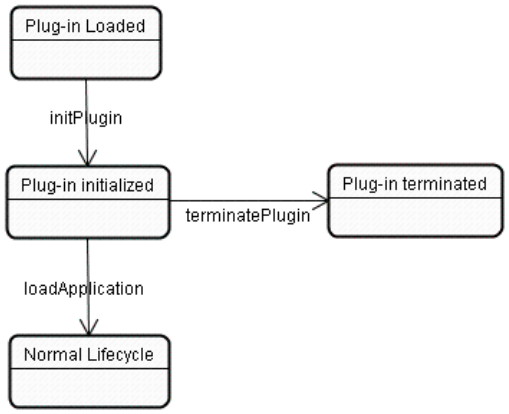

Ilustração 19 - Ciclo de Vida de Aplicações com Plug-in

Processamento de Dados:

Este componente trata da leitura e gravação de dados em arquivos de configuração do *Gerenciador de Aplicações*. Devem-se utilizar arquivos no formato *XML*. A *AIT* pode ser tanto uma classe java como um arquivo *XML* com a descrição das aplicações. A escolha é livre para o desenvolvedor do *Gerenciador de Aplicações*, exemplo: no *middleware* MHP se utiliza arquivos *XML*, no *middleware* do SBTVD usa-se uma classe.

Provider:

Este componente só deve ser incorporado ao *Gerenciador de Aplicações* caso haja infra-estrutura para isso. Através deste o *Gerenciador de Aplicações* dá aos terminais de acesso uma capacidade extra: o acesso remoto, podendo ser via celular, *PDA*, *Internet* ou até mesmo outro terminal de acesso.

Um *Provider* seria um servidor de aplicações. No caso, um *publisher* de aplicações na rede – isto só é possível porque cada terminal de acesso possui um endereço *"MAC"*[18] associado, bem como um endereço *IP*, o que permite sua identificação em uma rede digital.

Este *publisher* disponibilizaria não apenas acesso às aplicações residentes e *storage* – perceba que apenas as aplicações *storage* poderiam ser disponibilizadas para *download* –, mas serviços dos

[18] *Medium Access Control*

mais variados tipos, como: gravação de programação, compra de *pay-per-view*, acesso a dados pessoais e preferências etc.

Sua arquitetura deve ser baseada na de *web services* [20], provendo-se um servidor, *skeletons* e *stubs*. Este trabalho não tem como foco a especificação deste serviço, deixando-o para trabalhos futuros.

Rede:

Este componente provê acesso à rede. Na realidade, disponibiliza uma interface entre o *Gerenciador de Aplicações* e a *API* do *Canal de Interatividade*. Contudo, caso o *Módulo Inteligente* estiver ativado, poderá prover alguns serviços diferenciados, como: controle de acesso à rede, tunelamento de troca de mensagens, criptografia etc.

Útil:

Componentes que não se integram em nenhum outro se enquadram aqui. Pode ser um algoritmo, um manipulador de caracteres, rotinas que se repetem etc.

3.3. Evolução

O *Gerenciador de Aplicações* tem muito a evoluir. Sua evolução deve acompanhar a demanda por novas aplicações e a própria evolução do terminal de acesso. As possibilidades são inúmeras, principalmente no que se diz respeito à capacidade de execução de aplicações mais pesadas e complexas, além de suporte a outras linguagens que não Java.

Curto Prazo

***Auto-Update*:**

O *vendor* do sistema publicará um serviço de *auto-update*. Através deste, o *Gerenciador de Aplicações* carregará atualizações e *patches* do sistema. Esta atualização poderá ser agendada pelo usuário, como feita em intervalos regulares.

O *auto-update* é importante principalmente para correção de *bugs* existentes nas versões do sistema. Em decorrência destas atualizações o usuário sempre contará com um sistema integro e confiável.

Suporte a Aplicações Complexas:

O termo *"Aplicações Complexas"* é bem amplo. Entretanto, neste caso específico, refere-se a aplicações que se utilizam de muitos recursos, requerem muito processamento, renderizam

imagens e gráficos pesados, iniciam muitas *Threads*, acessam recursos remotos e processam diversos tipos de sons.

Pode-se concluir que jogos, programas de áudio e alguns programas de escritório poderão ser executados no *Gerenciador de Aplicações*.

MÉDIO PRAZO

Acesso Remoto:

Este recurso foi especificado nesta proposta, contudo ainda não há infra-estrutura necessária para sua real aplicação. Dentro de um médio prazo, poderá ser posto em prática, permitindo acesso remoto ao seu terminal de acesso.

Integração a Eletrodomésticos:

É possível integrar eletrodomésticos em uma rede a fim de se alcançar a chamada casa inteligente. Cada eletrodoméstico seria considerado uma aplicação remota, tendo seu ciclo de vida gerenciado pelo *Gerenciador de Aplicações* – há a necessidade de se estender o modelo de ciclo de vida existente.

Essa integração permitiria ao usuário controlar sua casa através da televisão, e a partir do acesso remoto, controlá-la remotamente. Cenários improváveis tornar-se-iam possíveis, como programar a temperatura da geladeira, travar as portas e janelas, acender e apagar luzes, além de movimentar câmeras de segurança, estando, por exemplo, em um cruzeiro de férias pelo atlântico.

LONGO PRAZO

***Peer-to-Peer*:**

O *Gerenciador de Aplicações,* fazendo parte de uma rede *P2P*, será capaz de solicitar e baixar *(download)* aplicações que estão em outros terminais de acesso, além de compartilhar as suas próprias aplicações. Com isso aplicativos como jogos ou outros mais complexos poderão ser distribuídos facilmente entre os usuários de TVDI.

Este recurso só estará disponível no momento em que as TVDIs, bem como sua infra-estrutura de rede, convergirem para algo próximo ao dos computadores atuais.

***Grid Computing*:**

Um *Gerenciador de Aplicações* será capaz de distribuir a execução de uma aplicação complexa com outros *Gerenciadores de Aplicações*; obtendo, assim, resultados combinados antes impossíveis de

ser alcançados isoladamente. Este recurso pode ser utilizado até por computadores, desde que o *Gerenciador de Aplicações* esteja conectado a uma rede, como a *Intenet* ou futuramente a *Internet 2*.

Este recurso permitiria um melhor aproveitamento do processamento de *Gerenciadores de Aplicações* ociosos – desde que autorizado pelos proprietários –, sendo os mesmos utilizados para a realização de diversas computações, onde é necessário um grande poder de processamento.

Gerenciar Aplicações em Outros *Gerenciadores de Aplicações*:

O *Gerenciador de Aplicações* estará apto a controlar o ciclo de vida de aplicações remotas, existentes apenas no contexto de outros *Gerenciadores de Aplicações*. Veja que este recurso é uma evolução de outro já apresentado nesta seção: Integração a Eletrodomésticos.

Este recurso permitirá a criação de aplicações ainda mais complexas, que precisem ser distribuídas, como no caso de alguns jogos *multiplayers*, *chats* (bate-papo) ou mesmo aplicações de vídeo-conferências e afins.

Capítulo IV

Implementação

"Alea Jacta Est"

- Julius Caesar

A partir do *Gerenciador de Aplicações* proposto no Capítulo III realizou-se a implementação de um subconjunto do mesmo, a fim de servir como prova de conceito. Vale ressaltar que este estudo faz parte de um esforço maior, que é a busca de um *Sistema Brasileiro de Televisão Digital (SBTVD).*

Como pesquisador membro do consórcio FlexTV – RFP-04, especificação de *middleware* de referência –, representante do Centro de Informática da UFPE, faço parte da equipe responsável pelo desenvolvimento do módulo *Gerenciador de Aplicações*. Muito dos conceitos aqui apresentados estão sendo desenvolvidos na prática.

4.1. Componentes Implementados

Dentro dos componentes propostos são aqui apresentados o *Controlador* e o componente interno responsável pelo gerenciamento do *Ciclo de Vida*. Foi criado também um *Parser XML* e uma *DTD*, partes integrantes do componente *Processamento de Dados,* através do qual se lê e se grava os dados das aplicações.

As assinaturas dos métodos relacionados a cada classe encontram-se no Apêndice IV.

4.1.1. Controlador

Como apresentado no Capítulo III este é o principal componente do *Gerenciador de Aplicações*. Dentro dele deve ser definido a *"Main Class"* do sistema. Convém-se denominar de **`ApplicationManager`**. Esta classe é um *singleton* devendo implementar a interface **`Runnable`** de Java, já que a mesma tem sua execução enquadrada no método ***`run()`***.

Esta é classe é responsável pelo carregamento de dados na **AIT**, que, por sua vez, também é uma classe Java. Nela estão listados todas as aplicações existentes, sendo através desta que o **ApplicationManager** acessa as informações de cada aplicação. Esta também é *singleton*.

Ainda na classe **ApplicationManager** é carregada uma instância da classe **WatchdogTimer** – responsável pelo gerenciamento das atividades do componente –, bem como uma instância de cada gerenciador, ou seja, são criados um **UserEventManager**, um **ComunicatorManager** e um **StorageManager**. Todas estas classes são *singletons*.

Voltando para a classe **AIT**, em sua instanciação, são criados os *proxys* de cada aplicação existente no terminal de acesso, sejam nativas ou *storage*. As aplicações nativas são também inicializadas, ou seja, um objeto **NativeContext** é repassado as aplicações, bem como uma referência para o ambiente Java, capacitando desde já a interação bi-direcional entre as aplicações nativas e o *Gerenciador de Aplicações*. Perceba que todo este carregamento de dados inicial é realizado através de um arquivo de configuração, escrito em *XML*.

Com o sistema devidamente configurado, o *Gerenciador de Aplicações* é ajustado como um *Thread Daemon*, na busca por liberar recursos.

4.1.2. Ciclo de Vida

Este componente baseou-se tanto na JavaTV API como no componente de gerenciamento de aplicações do DVB, contudo foi necessário implementar uma extensão dos mesmos, a fim de se conseguir dar suporte às aplicações nativas. O diagrama de classes é apresentado na Ilustração 20. É perceptível uma grande semelhança com a arquitetura apresentada no Capítulo III, contudo a que aqui se encontra apresentada, é apenas um subconjunto da mesma.

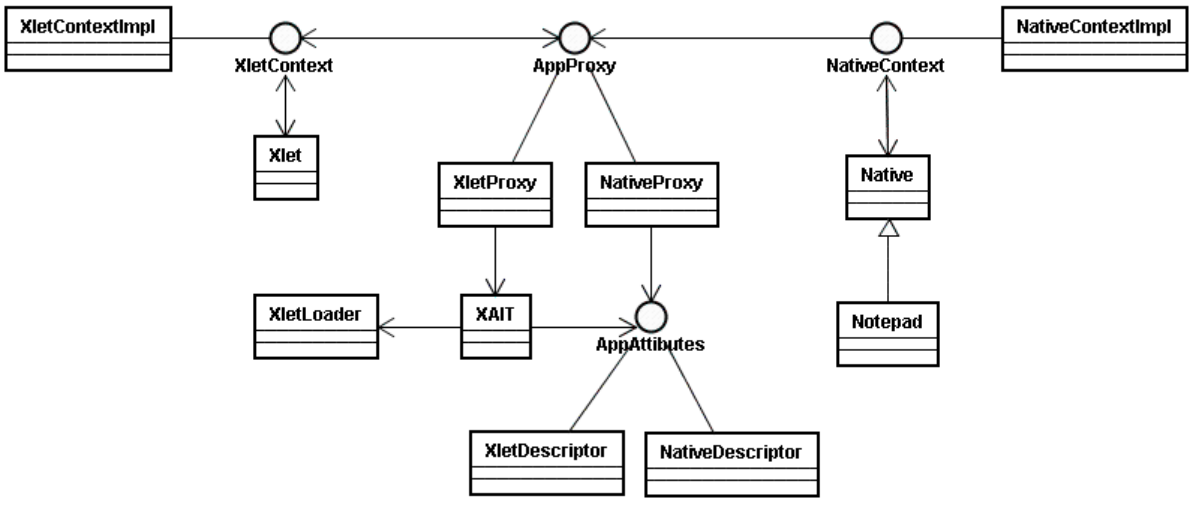

Ilustração 20 - Diagrama de Classes do Ciclo de Vida - Implementação

Fica evidente a importância da interface **AppProxy**, especificada pelo DVB. É a partir da extensão da mesma que especializamos o conceito geral de aplicação, surgindo conceitos específicos para *Xlets* e aplicações nativas, representadas no diagrama como **XletProxy**, **NativeProxy**. É de se ressaltar, ainda, a quebra de interface **AppAttributes** em **XletDescriptor** e **NativeDescriptor**, já que o regime destas aplicações é diferente os atributos também o são.

Outro fator que chama a atenção nesta modelagem é a classe **XAIT**, proposta por Steven Morris [21]. Ela concentra os dados necessários ao gerenciamento de *Xlets*. Deve-se implementar, no futuro, uma para as aplicações nativas, bem como para aplicações declarativas.

4.1.3. Processamento de Dados

Para este componente foi implementado um *Parser XML* e definida uma gramática para o mesmo. Utilizou-se a arquitetura *DOM* para manipular o *XML*. Assim foram elaboradas as classes **ApplicationManagerScanner** e **ApplicationManagerWriter**.

A gramática definida para esta implementação é bem simples dando suporte a aplicações nativas e a *Xlets* Java, como é apresentado a seguir.

```
<?xml version='1.0' encoding='UTF-8'?>
<!--
    An example how to use this DTD from your XML document:
    <?xml version="1.0"?>
    <!DOCTYPE applications SYSTEM "applicationManager.dtd">
    <applications>
    ...
    </applications>
-->
<!--- Put your DTDDoc comment here. -->
<!ELEMENT native-executable (#PCDATA)>
<!--- Put your DTDDoc comment here. -->
<!ELEMENT native-locator (#PCDATA)>
<!--- Put your DTDDoc comment here. -->
<!ELEMENT native-type (#PCDATA)>
<!--- Put your DTDDoc comment here. -->
<!ELEMENT native (netive-executable|netive-type|netive-locator)*>
<!--- Put your DTDDoc comment here. -->
<!ELEMENT netives (native)*>
<!--- Put your DTDDoc comment here. -->
<!ELEMENT xlet-locator (#PCDATA)>
<!--- Put your DTDDoc comment here. -->
<!ELEMENT xlet-oid (#PCDATA)>
```

```
<!--- Put your DTDDoc comment here. -->
<!ELEMENT xlet-aid (#PCDATA)>
<!--- Put your DTDDoc comment here. -->
<!ELEMENT xlet-appID (xlet-oid|xlet-aid)>
<!--- Put your DTDDoc comment here. -->
<!ELEMENT xlet (xlet-locator|xlet-appID)*>
<!--- Put your DTDDoc comment here. -->
<!ELEMENT xlets (xlet)*>
<!--- Put your DTDDoc comment here. -->
<!ELEMENT applications (residents|xlets)*>
```

Tabela 8 - Gramática Desenvolvida

4.2. Resultados

Conseguiu-se realizar com sucesso todo o ciclo de vida das aplicações, transitando por todos os estados. Foram simuladas notificações de chegada de várias aplicações seguindo à risca os fluxos principais e alternativos descritos no Apêndice III, bem como a execução de uma aplicação nativa – o *Notepad* do *Windows*.

Executou-se ainda a gravação de novas aplicações no arquivo de configuração do sistema, bem como a execução de *Xlets* mapeados neste mesmo arquivo, ou seja, aplicações *storage*.

4.2.1. Dificuldades

Durante o processo de desenvolvimento surgiram diversas dificuldades, sendo a principal delas o manuseio de *Threads*. Quando se executava apenas aplicações *Xlets*, ou seja, Java, a execução corria sem problemas de maneira limpa e correta. Entretanto, quando se executava uma aplicação nativa – no caso o *Notepad* – a máquina virtual perdia o controle sobre as *Threads Java* e sobre o processo nativo.

A solução adotada foi percorrer o código à procura de pontos onde se faria obrigatório o uso do modificador **`synchronized`** de Java. Este modificador cria um *lock* na área selecionada impedindo a execução interpolada. Serviu, todavia, apenas como paliativo, tendo-se posteriormente que inserir *"bolhas"* – paradas momentâneas – no processo de execução.

Capítulo V

Conclusões

"O progresso é a realização das Utopias."

- Oscar Wilde

Neste trabalho apresentamos e discutimos vários conceitos sobre serviços em televisão digital interativa. Foram introduzidos, em primeira instância, estudos sobre a transmissão do sinal de *broadcast*, o qual multiplexa os fluxos elementares de vídeo, de áudio e de dados. Também foi explicada a forma com que este sinal alcança os lares dos telespectadores e, onde ocorre a demultiplexação do mesmo: nos *set-top boxes* ou simplesmente terminais de acesso. Em seguida, foi realizada uma breve explanação sobre os padrões de TVDI abertos, focando-se nas suas divergências, arrematando com uma comparação entre os mesmos.

Finalmente foi apresentado um dos temas principais do trabalho, as aplicações, sendo analisadas detalhadamente. Foi então, apresentado as diversas categorias de aplicações junto com exemplos e dicas de desenvolvimento. Foi elaborada uma proposta para solucionar a portabilidade de aplicações desenvolvidas para diferentes plataformas: unificação da *Application Information Table (AIT)*, ou compactação de arquivos descritores de informação junto com as aplicações.

Foram apresentados, também, pontos de extensão na JavaTV API da *Sun Microsystems*. Ative-me, principalmente, na extensão do objeto de comunicação bi-direcional entre o sistema embarcado no terminal de acesso e as aplicações: a interface **`XletContext`**. Propus novos métodos que possibilitam acesso a dados da plataforma e, exemplifiquei o seu uso, através da necessidade de que uma determinada aplicação "A" tem em iniciar uma interação com uma outra "B". Foi apresentada, também, uma extensão de gramática para aplicações declarativas, sendo exposto no Capítulo II uma maneira de construir este tipo específico de aplicação.

O segundo foco deste trabalho, um *Gerenciador de Aplicações*, é então apresentado com um maior detalhamento. Realizou-se um *overview* sobre o estado da arte deste importante componente do *middleware* de televisão digital, sendo apresentada uma proposta de unificação das funcionalidades existentes, porém isoladas, em diferentes consórcios, bem como inovações.

Ao término dos capítulos principais foram apresentados cenários de evolução divididos em três intervalos de tempo: curto, médio e longo prazo.

Desta forma, pode-se concluir, no que concerne às aplicações, que em um primeiro momento os consórcios mundiais atêm-se as procedurais, desenvolvidas em Java, de simples implementação e de pouca carga gráfica, tendo, ainda, muito que se desenvolver no âmbito das aplicações declarativas. Conclui-se também que, de um modo geral, a evolução das aplicações é limitada pela evolução da plataforma na qual executam, e que para um efetivo uso das mesmas, fazem-se necessários ajustes e extensões às bibliotecas disponíveis no mercado.

Voltando-se para o *middleware*, em particular ao módulo proposto – o *Gerenciador de Aplicações* –, conclui-se que o mesmo é um sistema já mais maduro, mas com pouco poderio de execução, sendo factível sua extensão, a fim de prover melhores serviços. A proposta apresentada tem este objetivo, fundamentando-se em uma linha de evolução já ocorrida com os computadores.

5.1. Trabalhos Futuros

A pesquisa e as propostas aqui apresentadas devem ser consideradas introdutórias. Considero apenas como a ponta do *iceberg*, requerendo-se um maior aprofundamento no tema e em sua discussão.

Por conseguinte, alguns trabalhos futuros seriam majoritários, como: um maior detalhamento das aplicações, focando não apenas na concepção, elaboração e modelagem, mas também na implementação; no que se diz respeito ao *Gerenciador de Aplicações,* será crucial um estudo detalhado do *middleware* e de sua arquitetura visando uma integração total entre ambos, bem como de testes de campo.

A implementação de protótipos baseados nesta especificação, bem como em uma futura evolução desta, poderiam validar a proposta aqui apresentada. Este caminho poderia levar ao surgimento de um *framework* no âmbito de *Gerenciamento de Aplicações*.

5.2. Considerações Finais

No sonho de uma sociedade interligada e socialmente eqüitativa, a TV, por ser um meio de comunicação de massa – basta apenas o indicador que no Brasil a percentagem de lares com a caixinha mágica é maior que a de lares que dispõem de geladeiras –, tem papel de destaque, sendo, portanto, um mercado disputado com apetite por *players* dos mais diversos portes, inclusive governos.

Assim, é de suma importância dominar a arte de desenvolver aplicações para este mercado, bem como entender a estrutura da plataforma, a fim de se tirar o máximo de desempenho da mesma, bem como vislumbrar o rumo da evolução, no intuito de se manter sempre na vanguarda.

Referências

"Stand on the shoulders of giants."

- Isaac Newton

[01] FLEXTV. TV Digital: Análise das Alternativas Tecnológicas. Relatório em atendimento ao Requisito 4.1.3 – RFP04/2004 – MCT/FINEP, produto 1A, dezembro, 2004.

[02] FlexTv – Visão de Contexto do Middleware – 2005

[03] FlexTv – Arquitetura Conceitual do Middleware – 2005

[04] Introdução à Televisão Digital Interativa:Arquitetura, Protocolos, Padrões e Práticas

[05] B. Calder, J. Courtney, B. Foote, L. Kyrnitszke, D. Rivas, C. Saito, J. VanLoo, and T.Ye, "Java TV API Technical Overview," the Java TV API White paper, Version1.0,November14, 2000

[06] Peng, Chengyuan – "Digital Television Applications", Finland 2002

[07] Cunha A. Neto, Fernando - Entendendo Data Broadcasting em Plataformas de TV Digital, Trabalho de Graduação, CIN-UFPE, 2004

[08] The Digital Vídeo Broadcasting Project, http://www.dvb.org.

[09] Multimedia Home Platform Specificication, http://www.mhp.org/

[10] Advanced Television Systems Comitee, http://www.atsc.org.

[11] DTV Aplication Software Environment Specification, http://www.dase.org

[12] ISO. (1996a). "ISO/IEC 13818-1 - Information Technology – Generic Coding of Moving
Pictures and Associated Audio Information – Part 1: Systems (MPEG-2 Systems)",

1996.

[13] Java TV API, http://java.sun.com/products/javatv/

[14] Java TV API specification 1.0, http://java.sun.com/products/javatv/

[15] Louise B. Monteiro, Monique -Uma Proposta de Categorização para Aplicações de TV Digital, Trabalho de Graduação, CIN-UFPE 2004

[16] Aguiar Loureiro, Janine – Interfaces de programação para o desenvolvimento de aplicações para TV Digital, CIN-UFPE, 2004

[17] Peng, Chengyuan & Vuorimaa, Petri – "*A DIGITAL TELETEXT* SERVICE" – International Conference on Computer Graphics, Visualization and Computer Vision, Czech Republic, February 5 - 9, 2001

[18] GT Middleware. Disponível em <*http://www.rnp.br/pd/gts2004-2005/middleware.html*>. Acessado em Agosto de 2005.

[19] Textos Importantes – SET. Disponível em <*http://www.set.com.br/textos_tvdigital.htm*>. Acessado em Maio de 2005.

[20] Kreger, Heather, *"Web Services Conceptual Architecture (WSCA 1.0)*" – IBM, Maio 2001.

[21] Morris, Steven, Smith-Chaigneau, Anthony – "Interactive TV Standards: A Guide to MHP, OCAP and JavaTV" –Elsevier, 2005

[22] A standard core specification for networking digital AV appliances. Disponível em <*http://www.havi.org/home.html*> Acessado em em Agosto de 2005.

APÊNDICE I

UM XLET SIMPLES

Neste apêndice apresento o código – baseado em [21], bem como o que deve ser feito em cada um dos métodos de uma aplicação *Xlet*. Este exemplo é um HelloWolrd. Os comentários contêm as explicações de cada uma das ações tomadas.

HELLOWORLDXLET

```
/**
 * Um Xlet simples para demonstração de como se desenvolver aplicações
 * para Televisão Digital Interativa
 *
 * @author Andrino S. S. Coêlho
 * @see javax.tv.xlet.Xlet
 */

// Deve-se importar as classes existentes no pacote javax.tv.xlet
import javax.tv.xlet.Xlet;
import javax.tv.xlet.XletContext;
import javax.tv.xlet.XletStateChangeException;

public class HelloWorldXlet implements javax.tv.xlet.Xlet {

      // Objeto que permitirá a comunição bi-lateral entre
      // sistema e a aplicação.
      private javax.tv.xlet.XletContext context;

      // Booleano indicativo de que a aplicação já
      // foi executada ao mínimo uma vez neste ciclo
      // de vida atual.
      private boolean hasBeenStarted;

      /*
       * Construtor Padrão
       * Deve ser definido para uso do Gerenciador de Aplicações
       * Não se deve inicializar nada neste método
       */
      public HelloWorldXlet () {}

      /*
       * Toda e qualquer inicialização deve ser feita
       * neste método, portanto inicie variáveis,
       * carregue classes e drivers.
       *
```

```
     *
     * @param   context     O objeto XletContext passado
     *                            pelo Gerenciador de Aplicações
     *
     * @throws  javax.xlet.XletStateChangeException  Caso não
     *                seja possível incializar, lance uma
     *                javax.xlet.XletStateChangeException avisando
     *                que o Xlet não pôde ser inicializado
     *                corretamente.
     */
    public void initXlet(javax.xlet.XletContext context)
          throws javax.xlet.XletStateChangeException {

          // Guarde o objeto context para uso posterior
          this.context = context;
          // Inicialize o booleano que indicará se a aplicação
          // já foi colocada em execução.
          // Será usado no método startXlet.
          this.hasBeenStarted = false;

          System.out.println("SimpleXlet foi inicializado com sucesso!");
    }

    /*
     * Ainda não ficou claro o que deve ser feito na
     * chamada deste método (Nem no MHP, nem no OCAP).
     *
     * Contudo as seguintes ações devem ser tomadas:
     * <li>Liberar qualquer recurso compartilhado que esteja usando</li>
     * <li>Parar qualquer thread não necessário</li>
     * <li>Retirar-se da Tela</li>
     *
     * @throws  javax.xlet.XletStateChangeException  Caso não
     *                seja possível incializar, lance uma
     *                javax.xlet.XletStateChangeException avisando
     *                que o Xlet não pôde ser pausado
     *                corretamente.
     */
    public void pauseXlet()
          throws javax.xlet.XletStateChangeException {
          System.out.println("HelloWorldXlet foi pausado com sucesso!");
    }

    /*
     * Executa o Xlet. O mesmo pode se apresentar na tela para
     * o usuário interagir com o mesmo.
     *
     * Contudo este método possui uma cilada para o sistema, já
     * que o caller tem de se esperar o retorno. Isso traz problemas
     * de concorrência, para evitá-los é recomendado que os Xlets
     * iniciem uma Thread que contenha a execução de suas principais
     * funcionalidades.
     *
     * @throws  javax.xlet.XletStateChangeException  Caso não
     *                seja possível incializar, lance uma
     *                javax.xlet.XletStateChangeException avisando
     *                que o Xlet não pôde ser pausado
     *                corretamente.
     */
    public void startXlet()
          throws javax.xlet.XletStateChangeException {
```

```
            if (this.hasBeenStarted) {
                  System.out.println("O Xlet está sendo re-executado");
                  System.out.println("HelloWorld”);
            } else {
                  System.out.println("O Xlet está em execução!");
                  System.out.println("HelloWorld”);
                  this.hasBeenStarted = true;
            }
      }

      /*
       * Para o Xlet. O parâmetro booleano diz ao Xlet se há uma
       * possibilidade de continuar em execução ou não. Caso seja
       * false o Xlet deve lançar um XletStateChangeException
       * indicando a vontade contrária do seu término.
       *
       * @param   forced      Valor booleano indicativo de destruição
       *                            forçada ou não.
       */
      public void destroyXlet(boolean forced)
            throws javax.xlet.XletStateChangeException {
            if (forced) {
                  System.out.println("Xlet forçado a se destruir!");
            } else
                  throw new javax.xlet.XletStateChangeException(
                        "Por favor! Não me mate! :p");
      }
}
```

Apêndice II

Construindo Aplicações em HTML

Neste apêndice, baseado no capítulo homônimo descrito por Steven Morris [21], apresento algumas poucas diretrizes de como desenvolver aplicações declarativas fazendo uso da linguagem HTML e associados. Todo o processo aqui descrito é voltado para o padrão DVB-HTML.

1. DVB-HTML

Existem, atualmente, diversas instituições desenvolvendo aplicações para o DVB-MHP. Muitas crêem ser mais interessante programá-las em linguagens declarativas, seja por uma maior facilidade, seja porque trabalham com esta tecnologia a bastante tempo. Por conseguinte, o consórcio do DVB introduziu na versão 1.1 do MHP as chamadas *DVB-HTML Applications*, a fim de dar suporte a aplicações declarativas descritas em diversos padrões de *Internet*, tais como: XHTML[19] 1.0, CSS[20] level 2, ECMAScript e DOM level 2.

Para se desenvolver aplicações *DVB-HTML* o desenvolvedor necessita especificar os limites de sua aplicação. Isto é feito através do uso de expressões regulares armazenadas em um arquivo sinalizado na *AIT* – campo *Application Boundary Descriptor*. Sendo obrigatório a inclusão deste arquivo no *Object Caroussel*.

Realizando um paralelo com aplicações *LINUX*[21], o que o desenvolvedor precisa é notificar ao *Gerenciador de Aplicações* quais são as páginas que fazem parte da aplicação como um todo.

2. Desenvolvendo

O desenrolar do desenvolvimento de aplicações DVB-*HTML* segue muito similar ao desenvolvimento de aplicações *Web* tradicionais.

---

[19] *Extended Hypertext Markup Language*
[20] *Cascade Style Sheet*
[21] Sistema Operacional de código aberto. O maior opositor, atualmente, do *Windows*.

2.1. Navegação

O desenvolvedor deve ter sempre em mente que as aplicações irão executar em um ambiente diferente do tradicional computador. Logo se deve evitar *"páginas"* que necessitem *scrolling*, seja vertical ou lateral, sendo estimulado o uso de *links* de navegação explícitos.

A fim de promover o *scrolling,* o DVB define propriedades adicionais em seu *dvb-tv media type*: nav-up, nav-down, nav-left, nav-right. Estas propriedades permitem que o desenvolvedor mapeie as teclas direcionais do controle remoto realizando o *scrolling* nas direções indicadas.

2.2. Apresentando na Tela

Assim como as aplicações procedurais Java, as DVB-HTML também devem utilizar uma instancia de **`HScene`** para adicionar seu conteúdo na tela. Todavia esta não será obtida através de **`HSceneFactory`**, mas através de uma diretiva inclusa no documento DVB-HTML: **`@dvb-viewport`**.

A diretiva **`@dvb-viewport`** permite que a aplicação requisite uma área da tela para apresentar-se, bem como uma grande variedade de propriedades, tais como: *scene, horizontal-resolution, vertical-resolution, initial, area, type, background-image-rectangle, background, background-video-rectangle, background-video-clip, background-video-preserve-aspect, background-video, background-video-transform* ect.

2.3. Captura de Eventos

Os eventos associados a aplicações declarativas são denominados de *triggers*, Desta forma o *Gerenciador de Aplicações* deve estar ciente da possibilidade de receber notificações através de *triggers* que podem representar uma mudança de estado da aplicação como um simples *clique* em um *link* ou botão de ação.

As aplicações DVB-HTML escutam apenas um subconjunto dos eventos DOM level 2: **`UIEvent`**, **`MutationEvent`** e **`DVBLifecycleEvent`**.

Ciclo de Vida das Application DVB-HTML

No Capítulo II não foi apresentado um ciclo de vida para as aplicações declarativas pelo fato de existirem tantos padrões em que pode-se desenvolver as mesmas. Cada padrão possui peculiaridades que tornam praticamente impossível uma generalização do processo. Assim torna-

se bem mais interessante apresenta um ciclo de vida já definido para um conjunto de padrões de *Internet,* como foi feito pelo DVB.

A cada mudança de estado é lançado um **`HTMLEvent`** indicando de qual estado uma aplicação está saindo e para qual estado está transitando. Igualmente como no XletStateChangeEvent, é encapsulado se a transição ocorreu normalmente e qual aplicação trnata realizar a mudança de estado.

Os estados possíveis de serem assumidos são: *DESTROYED, KILLED, LOADING, NOT_LOADED, PAUSED, STARTED*; os eventos estão descritos na Tabela 1Tabela 9.

Evento	Descrição
AppStarting	Aplicação recebeu o evento **`dvb.start`**. Gerado apenas quando uma aplicação é sinalizada como PRFETCH na *AIT*
AppActive	Lançado na transição entre o estado *LOADING* e o estado *ACTIVE*.
AppPause	Lançado na transição entre o estado *ACTIVE* e o estado *PAUSED*.
AppResume	Lançado na transição entre o estado *PAUSED* e o estado *ACTIVE*.
AppDestroyed	Lançado na transição entre qualquer estado e o estado *DESTROYED*.
AppKilled	Lançado na transição entre qualquer estado e o estado *KILLED*.
AppTerminating	Lançado qunado uma aplicação em um *sub-frame* é terminada.

Tabela 9 - Eventos do Ciclo de Vida de uma Application DVB-HTML

2.4. Acesso à *Scripts*

É praticamente impossível, nos dias atuais, pensar-se em aplicações Web sem scripts de validação. Como aplicações DVB-HTML são similares à estas, incluiu-se, também, um padrão aberto de *script*, o: *ECMAScript*, o qual lembra em muitos aspectos a *JavaScript*, sendo compatível até a versão 1.3 do mesmo.

Não obstante, a inclusão de *ECMAScript* não se deve apenas a validação, mas tem como foco principal a inclusão de ações procedurais Java em aplicativos DVB-HTML, sem que haja a necessidade de se manipular aplicações internas. Isto é possível através do **`package Packages`**, incluso no MHP 1.1.

2.5. Aplicações Internas

O Capítulo II explana a existência de aplicações internas, citando justamente o uso junto com aplicações declarativas, sendo apresentado uma extensão da gramática proposta por

Chengyuan Peng e Petr Vuorimaa [17] para a inclusão de um novo elemento que dê suporte às mesmas.

3. Considerações Finais

Como já foi explanado neste trabalho, o suporte a este tipo de aplicação é opcional não havendo atualmente nenhum terminal de acesso que dê suporte completo a todas as funcionalidades das aplicações DVB-HTML.

Isso não significa que seu estudo seja relegado a um segundo plano, mas apenas que os conceitos envolvidos estão na vanguarda da tecnologia aplicada no desenvolvimento de aplicações para a TVDI. Entretanto, é fato que havendo com um aumento no mercado pela demanda destas aplicações rapidamente serão desenvolvidos terminais que dêem suporte às mesmas.

Apêndice III

Detalhamento dos Casos de Uso

Neste apêndice encontram-se relatados de maneira detalhada os casos de uso do módulo *Gerenciador de Aplicações* descrito neste Trabalho de Graduação.

1. Definições e Acrônimos

1.1. Definições

A correta interpretação deste apêndice exige o conhecimento de algumas convenções e termos específicos, que são descritos a seguir.

1.2. Identificação dos Requisitos

Por convenção, a referência a requisitos é feita através do identificador do requisito, de acordo com a especificação a seguir:

[identificador do requisito]

Os requisitos devem ser identificados com um identificador único. A numeração inicia com o identificador [RF-01] ou [NF-01] e prossegue sendo incrementada à medida que forem surgindo novos requisitos.

1.3. Prioridades dos Requisitos

Para estabelecer a prioridade dos requisitos, foram adotadas as denominações "essencial", "importante" e "desejável".

- Essencial é o requisito sem o qual o sistema não entra em funcionamento. Requisitos essenciais são requisitos imprescindíveis, que têm que ser implementados impreterivelmente.
- Importante é o requisito sem o qual o sistema entra em funcionamento, mas de forma não satisfatória. Requisitos importantes devem ser implementados, mas, se não forem, o sistema poderá ser implantado e usado mesmo assim.

- Desejável é o requisito que não compromete as funcionalidades básicas do sistema, isto é, o sistema pode funcionar de forma satisfatória sem ele. Requisitos desejáveis podem ser deixados para versões posteriores do sistema, caso não haja tempo hábil para implementá-los na versão que está sendo especificada.

2. Requisitos Funcionais

[RF - 01] - Carregar uma Aplicação	
O *Gerenciador de Aplicações* pode ser requisitado a carregar uma aplicação existente no sistema. É atribuído um *Loader* à mesma.	
Atores	Aplicação/*Gerenciador de Aplicações*
Prioridade	Essencial
Entradas e pré-condições	• A Aplicação deve existir no terminal de acesso • A Aplicação deve encontrar-se no estado *NOT_LOADED*. • Deve haver recursos disponíveis à criação do *Loader* • Aplicação solicitante deve ter permissão para carregar outra aplicação *(AUTOSTART)*
Saídas e pós-condições	• Um *Loader* é atribuído à aplicação • O estado da aplicação modifica-se de *NOT_LOADED* para *LOADED*.
Fluxos de Eventos	
Fluxo Principal	1. O *Gerenciador de Aplicações* recebe uma requisição de carregar uma aplicação. 2. O *Gerenciador de Aplicações* verifica que a aplicação encontra-se em um estado válido. 3. O *Gerenciador de Aplicações* instancia um *Loader* para a aplicação requisitada. 4. O *Gerenciador de Aplicações* solicita ao *Proxy* da aplicação requisitada que passe para o estado LOADED. 5. O *Proxy* da aplicação requisitada dispara um evento de mudança de estado *(XletStateChange)* para todos os *listeners* cadastrados nele.
Fluxo Excepcional	1. No passo 1 do fluxo principal, a requisição é feita por uma aplicação sem permissão de acessibilidade. 2. Uma *SecurityException* é lançada. 3. Uma Mensagem de erro é apresentada na tela (dispositivo de saída) do sistema.
Fluxo Excepcional 2	1. No passo 2 do fluxo principal a aplicação requisitada encontra-se em estado inválido. 2. Um *XletStateChangeException* é lançada. 3. Uma Mensagem de erro é apresentada na tela (dispositivo de saída) do sistema.

[RF – 02] – Inicializar uma Aplicação	
O *Gerenciador de Aplicações* pode ser requisitado a inicializar uma aplicação existente no sistema. É passado à aplicação um objeto simbolizando o Contexto da plataforma. Através do Contexto será realizada a comunicação entre a aplicação e o *Gerenciador de Aplicações*.	
Atores	Aplicação/*Gerenciador de Aplicações*
Prioridade	Essencial
Entradas e pré-condições	• A Aplicação deve existir no *Gerenciador de Aplicações* • A Aplicação deve encontrar-se no estado *NOT_LOADED* ou *LOADED* • Aplicação solicitante deve ter permissão para incializar outra aplicação *(AUTOSTART)*
Saídas e pós-condições	• A Aplicação é inicializada • Um objeto Contexto é passado à Aplicação • O estado da aplicação modifica-se de *NOT_LOADED* ou *LOADED* para *PAUSED*.
Fluxos de Eventos	
Fluxo Principal	1. O *Gerenciador de Aplicações* recebe uma requisição de inicializar uma aplicação. 2. O *Gerenciador de Aplicações* verifica que a aplicação encontra-se em um estado válido. 3. O *Gerenciador de Aplicações* instancia um *Contexto*. 4. O *Gerenciador de Aplicações* instancia a aplicação requisitada. 5. O *Gerenciador de Aplicações* repassa o *Contexto* para a aplicação requisitada. Este passo é conhecido como *initXlet*. 6. O *Gerenciador de Aplicações* solicita ao *Proxy* da aplicação requisitada que passe para o estado *PAUSED*. 7. O *Proxy* da aplicação requisitada dispara um evento de mudança de estado *(XletStateChange)* para todos os *listeners* cadastrados nele.
Fluxo Excepcional 1	1. No passo 1 do fluxo principal, a requisição é feita por uma aplicação sem permissão de acessibilidade. 2. Uma *SecurityException* é lançada. 3. Uma Mensagem de erro é apresentada na tela (dispositivo de saída) do sistema.
Fluxo Excepcional 2	4. No passo 2 do fluxo principal a aplicação requisitada encontra-se em estado inválido. 5. Um *XletStateChangeException* é lançada. 6. Uma Mensagem de erro é apresentada na tela (dispositivo de saída) do sistema.

[RF - 03] – Executar uma Aplicação	
O *Gerenciador de Aplicações* pode ser requisitado a por em execução uma aplicação existente no sistema. A aplicação é colocada no estado ativo e passa a disputar com outras aplicações o *pool* de recursos compartilhados do sistema.	
Atores	Usuário TVDI/Aplicação/*Gerenciador de Aplicações*
Prioridade	Essencial
Entradas e pré-condições	• A Aplicação deve existir no *Gerenciador de Aplicações* • A Aplicação deve encontrar-se no estado *NOT_LOADED*, *LOADED* ou *PAUSED*. • Deve haver recursos disponíveis à execução da Aplicação • Aplicação solicitante deve ter permissão para executar outra aplicação *(AUTOSTART)*
Saídas e pós-condições	• A Aplicação é colocada em execução • O pool de recursos compartilhados do sistema é disponibilizado à aplicação • A Aplicação é passada para o estado STARTED
Fluxos de Eventos	
Fluxo Principal	1. O *Gerenciador de Aplicações* recebe uma requisição de colocar em execução uma aplicação. 2. O *Gerenciador de Aplicações* verifica que a aplicação encontra-se em um estado válido. 3. O *Gerenciador de Aplicações* instancia uma *Thread*. 4. O *Gerenciador de Aplicações* repassa à *Thread* a aplicação requisitada. 5. O *Gerenciador de Aplicações* põe a *Thread* em execução. 6. O *Gerenciador de Aplicações* solicita ao *Proxy* da aplicação requisitada que passe para o estado *STARTED*. 7. O *Proxy* da aplicação requisitada dispara um evento de mudança de estado *(XletStateChange)* para todos os *listeners* cadastrados nele.
Fluxo Excepcional 1	1. No passo 1 do fluxo principal, a requisição é feita por uma aplicação sem permissão de acessibilidade. 2. Uma *SecurityException* é lançada. 3. Uma Mensagem de erro é apresentada na tela (dispositivo de saída) do sistema.
Fluxo Excepcional 2	1. No passo 2 do fluxo principal a aplicação requisitada encontra-se em estado inválido. 2. Um *XletStateChangeException* é lançada. 3. Uma Mensagem de erro é apresentada na tela (dispositivo de saída) do sistema.

[RF - 04] –Parar uma Aplicação	
O *Gerenciador de Aplicações* pode ser requisitado a parar uma aplicação existente no sistema. A aplicação tem sua execução parada ou, dependendo da mesma ser vinculada a um serviço (programa), é destruída e apagada do sistema.	
Atores	Usuário TVDI/Aplicação/*Gerenciador de Aplicações*
Prioridade	Essencial
Entradas e pré-condições	• A Aplicação pode estar em qualquer estado. • A Aplicação solicitante deve ter permissão para parar outra aplicação *(AUTOSTART)*
Saídas e pós-condições	• A Aplicação passa ao estado de *DESTROYED* – se vinculada a um serviço, ou ao estado *NOT_LOADED* caso contrário. • O objeto contexto é removido da Aplicação • A Aplicação perde o acesso ao pool de recursos compartilhados do sistema
Fluxos de Eventos	
Fluxo Principal	1. O *Gerenciador de Aplicações* recebe uma requisição de parar a execução de uma aplicação. 2. O *Gerenciador de Aplicações* verifica que a aplicação encontra-se em um estado válido. 3. O *Gerenciador de Aplicações* destrói a *Thread* vinculada à aplicação. 4. O *Gerenciador de Aplicações* destrói o objeto *Contexto* passado à aplicação. 5. O *Gerenciador de Aplicações* verifica que a aplicação não é vinculada a um serviço. 6. O *Gerenciador de Aplicações* solicita ao *Proxy* da aplicação requisitada que passe para o estado *NOT_LOADED*. 7. O Proxy da aplicação requisitada dispara um evento de mudança de estado *(XletStateChange)* para todos os *listeners* cadastrados nele. 8. O *Gerenciador de Aplicações* destrói o *Loader* associado à aplicação. *9*. A configuração *"Módulo Inteligente"* do *Gerenciador de Aplicações* está desativada.
Fluxo Alternativo 1	1. No passo 5 do fluxo principal, a aplicação é vinculada a um serviço. 2. O *Gerenciador de Aplicações* solicita ao *Proxy* da aplicação requisitada que passe para o estado *DESTROYED*. 3. O *Proxy* da aplicação requisitada dispara um evento de mudança de estado *(XletStateChange)* para todos os *listeners* cadastrados nele. 4. O *Gerenciador de Aplicações* destrói o *Loader* associado à aplicação. 5. O *Gerenciador de Aplicações* destrói o *Proxy* associado à aplicação. 6. O *Gerenciador de Aplicações* exclui a aplicação do sistema.
Fluxo Alternativo 2	1. No passo 9 do fluxo principal, a configuração "Módulo Inteligente" do *Gerenciador de Aplicações* está ativada. 2. O *Gerenciador de Aplicações* verifica que há aplicações na pilha de

	aplicações. 3. *Extends* [RF – 03]
Fluxo Alternativo 3	1. No passo 2 do fluxo alternativo 2, não há aplicações na pilha. 2. O passo 3 do fluxo alternativo 2 é ignorado.
Fluxo Excepcional 1	1. No passo 1 do fluxo principal, a requisição é feita por uma aplicação sem permissão de acessibilidade. 2. Uma *SecurityException* é lançada. 3. Uma Mensagem de erro é apresentada na tela (dispositivo de saída) do sistema.
Fluxo Excepcional 2	1. No passo 2 do fluxo principal a aplicação requisitada encontra-se em estado inválido. 2. Um *XletStateChangeException* é lançada. 3. Uma Mensagem de erro é apresentada na tela (dispositivo de saída) do sistema.

[RF - 05] – Pausar uma Aplicação	
O *Gerenciador de Aplicações* pode ser requisitado a pausar uma aplicação em execução no sistema. A aplicação é pausada e, no caso de estar com foco, o perde.	
Atores	Aplicação/*Gerenciador de Aplicações*
Prioridade	Essencial
Entradas e pré-condições	• A Aplicação encontra-se no estado *STARTED* • A Aplicação solicitante deve ter permissão para pausar outra aplicação *(AUTOSTART)*
Saídas e pós-condições	• A Aplicação passa do estado *STARTED* para o estado *PAUSED* • A Aplicação perde o acesso ao *pool* de recursos compartilhados do sistema
Fluxos de Eventos	
Fluxo Principal	1. O *Gerenciador de Aplicações* recebe uma requisição de pausar a execução de uma aplicação. 2. O *Gerenciador de Aplicações* verifica que a aplicação encontra-se em um estado válido. 3. O *Gerenciador de Aplicações* emite um sinal para aplicação solicitando sua parada momentânea. 4. O *Gerenciador de Aplicações* retira a aplicação do grupo que tem acesso aos recursos compartilhados. 5. O *Gerenciador de Aplicações* requisita perda do foco para a aplicação. 6. O *Gerenciador de Aplicações* solicita ao *Proxy* da aplicação requisitada que passe para o estado *PAUSED*. 7. O *Proxy* da aplicação requisitada dispara um evento de mudança de estado *(XletStateChange)* para todos os *listeners* cadastrados nele.
Fluxo Alternativo	1. No passo 1 do fluxo principal, a requisição é feita pelo próprio

	Gerenciador de Aplicações. 2. A configuração *"Módulo Inteligente"* do *Gerenciador de Aplicações* está ativada. 3. O *Gerenciador de Aplicações* empilha a aplicação. 4. Retoma o fluxo principal no passo 5.
Fluxo Excepcional 1	5. No passo 1 do fluxo principal, a requisição é feita por uma aplicação sem permissão de acessibilidade. 6. Uma *SecurityException* é lançada. 7. Uma Mensagem de erro é apresentada na tela (dispositivo de saída) do sistema.
Fluxo Excepcional 2	1. No passo 2 do fluxo principal a aplicação requisitada encontra-se em estado inválido. 2. Um *XletStateChangeException* é lançada. 3. Uma Mensagem de erro é apresentada na tela (dispositivo de saída) do sistema.

[RF - 06] - Limpar Tela *(CLEAR)*	
O *Gerenciador de Aplicações* pode ser requisitado a limpar a tela (dispositivo de saída) do sistema, parando a execução de qualquer aplicação vigente.	
Atores	Usuário
Prioridade	Importante
Entradas e pré-condições	• Não se aplica
Saídas e pós-condições	• A tela (dispositivo de saída) do sistema mostrará apenas o vídeo capturado pelo do *broadcast*
Fluxos de Eventos	
Fluxo Principal	1. Usuário pressiona botão *CLEAR* no controle remoto. 2. O *Gerenciador de Aplicações* "limpa" a tela (dispositivo de saída) do sistema, mostrando apenas o vídeo capturado pelo *broadcast*.

[RF - 07] – Receber Notificação de Download de Aplicação	
O *Gerenciador de Aplicações* é notificado, pelo módulo de processamento de dados do *middleware*, do término do *download* de uma aplicação.	
Atores	Módulo de Processamento de Dados do *Middleware*
Prioridade	Essencial
Entradas e pré-condições	• O *download* da aplicação deve te terminado
Saídas e pós-condições	• A aplicação recém *downloaded* está em execução.
Fluxos de Eventos	
Fluxo Principal	1. O módulo de Processamento de Dados do *middleware* notifica o *Gerenciador de Aplicações* da conclusão do *download* de uma aplicação.

	2. O *Gerenciador de Aplicações* verifica que a mesma é *AUTOSTART*. 3. O *Gerenciador de Aplicações* verifica que o usuário dá permissão de execução às aplicações *donwnloadable*. 4. O *Gerenciador de Aplicações* verifica se a aplicação pode ser executada sobre a plataforma do sistema – *Check Profile*. 5. *Include* [RF-01] 6. *Include* [RF-02] 7. *Include* [RF-03] 8. O *Gerenciador de Aplicações* inicia o gerenciamento da execução da aplicação através de um Cão de Guarda.
Fluxo Alternativo 1	1. No passo 2 do fluxo principal, não se trata de uma aplicação *AUTOSTART*. 2. *Extends* [RF-08]
Fluxo Alternativo 2	1. No passo 3 do fluxo principal, a execução de aplicações encontra-se BLOQUEADO. 2. O *Gerenciador de Aplicações* verifica que a notificação de aplicações encontra-se habilitado. 3. Um ícone é apresentado como notificação da chegada de uma aplicação. 4. Informações sobre a aplicação são mostradas na tela (dispositivo de saída) do sistema. 5. O foco é entregue ao ícone. 6. Usuário pressiona OK ou MENU. 7. Retorna ao passo 4 do fluxo principal.
Fluxo Alternativo 3	1. No passo 6 do fluxo alternativo 2, o usuário pressiona CLEAR. 2. A aplicação é excluída.
Fluxo Alternativo 4	1. No passo 2 do fluxo alternativo 2, a notificação de aplicações encontra-se desabilitada. 2. A configuração "Módulo Inteligente" do Gerenciador de Aplicação está ativada. 3. O *Gerenciador de Aplicações* verifica que a aplicação é vinculada a um serviço. 4. O *Gerenciador de Aplicações* verifica o tempo de duração do serviço. 5. O *Gerenciador de Aplicações* marca a aplicação com um temporizador vinculado ao término do serviço. 6. O *Gerenciador de Aplicações* guarda a aplicação numa pilha especial. 7. O serviço termina. 8. O *Gerenciador de Aplicações* verifica que o usuário não modificou a permissão de execução de aplicações. 9. Aplicação é excluída.
Fluxo Alternativo 5	1. No passo 2 do fluxo alternativo 4, a configuração "Módulo Inteligente" está desativada.

	2. O *Gerenciador de Aplicações* exclui a aplicação do sistema.
Fluxo Alternativo 6	1. No passo 3 do fluxo alternativo 4, a aplicação não é vinculada a um serviço. 2. O *Gerenciador de Aplicações* exclui a aplicação do sistema (Segurança – pode ser um vírus!).
Fluxo Alternativo 7	1. No passo 8 do fluxo alternativo 4 o usuário modificou a permissão de execução de aplicações. 2. A aplicação é retirada da pilha e retoma a seqüência do fluxo principal partindo do passo 4.
Fluxo Alternativo 8	1. No passo 4 do fluxo principal, a aplicação não está adaptada ao perfil do sistema. 2. *Extends* [RF-16]

[RF – 08] – Salvar uma Aplicação	
O *Gerenciador de Aplicações* é notificado, pelo módulo de processamento de dados do *middleware*, do *download* de uma aplicação, questionando o usuário se deseja armazená-la para uma posterior execução.	
Atores	Usuário TVDI
Prioridade	Essencial
Entradas e pré-condições	• O *download* da aplicação deve te terminado
Saídas e pós-condições	• É apresentada na tela (dispositivo de saída) do sistema uma janela intuitiva de armazenamento, ou não, de uma aplicação
Fluxos de Eventos	
Fluxo Principal	1. Uma mensagem é exibida na tela, perguntando se o usuário deseja realizar salvar a aplicação para posterior execução. 2. Usuário concorda com o armazenamento. 3. O *Gerenciador de Aplicações* verifica se há recursos suficientes para salvar a aplicação 4. A aplicação é salva. 5. A aplicação é incluída na lista de aplicações exibida pelo *Gerenciador de Aplicações*.
Fluxo Alternativo	1. No passo 2 do fluxo principal, caso o usuário não concorde com a instalação da aplicação, o sistema a remove do sistema.
Fluxo Excepcional	1. No passo 3 do fluxo principal, caso não haja recursos suficientes para a instalação o *Gerenciador de Aplicações* informará explicitamente ao usuário que o sistema não pôde efetuar a operação.

[RF – 09] – Excluir uma Aplicação	
O *Gerenciador de Aplicações* pode ser requisitado pelo usuário para excluir uma aplicação existente no sistema.	
Atores	Usuário

Prioridade	Essencial
Entradas e pré-condições	• A Aplicação deve existir no sistema
Saídas e pós-condições	• A Aplicação é excluída do sistema • A *AIT* é atualizada
Fluxos de Eventos	
Fluxo Principal	1. Usuário seleciona na interface do *Gerenciador de Aplicações* a aplicação desejada. 2. Uma mensagem é exibida na tela (dispositivo de saída) pedindo a confirmação ao usuário da remoção da aplicação. 3. Usuário concorda com a exclusão. 4. A aplicação é excluída do sistema. 5. A aplicação é excluída da lista de aplicações exibida pelo *Gerenciador de Aplicações* 6. A *AIT* é atualizada.
Fluxo Alternativo	1. No passo 3 do fluxo principal, caso o usuário não concorde com a exclusão da aplicação, nada é feito pelo sistema.
Fluxo Excepcional	1. No passo 4 do fluxo principal, caso a aplicação esteja em execução a operação é cancelada e o usuário notificado do fato

[RF – 10] – Navegar nas Aplicações	
O *Gerenciador de Aplicações* permitirá que o usuário navegue nas aplicações existentes no sistema.	
Atores	Usuário
Prioridade	Essencial
Entradas e pré-condições	• As Aplicações devem existir no sistema.
Saídas e pós-condições	• Não se aplica
Fluxos de Eventos	
Fluxo Principal	1. Usuário pressiona o botão do controle remoto correspondente ao Navegador de Aplicações. 2. Sistema exibe uma tela com as aplicações existentes no dispositivo. 3. O usuário navega nas aplicações pressionando as teclas direcionais do controle remoto
Fluxo Alternativo	1. No passo 3 do fluxo principal, caso a opção de "Exibir as Informações da Aplicação" esteja ativada, informações referentes às aplicações serão exibidas durante a navegação.

[RF – 11] – Visualizar Informações da Aplicação	
O *Gerenciador de Aplicações* deve permitir ao usuário obter informações sobre a aplicação selecionada.	
Atores	Usuário

Prioridade	Desejável
Entradas e pré-condições	• A Aplicação deve existir no sistema • A Aplicação deve possuir informações associadas
Saídas e pós-condições	• As informações associadas à aplicação selecionada serão apresentadas na tela (dispositivo de saída)
Fluxos de Eventos	
Fluxo Principal	1. Ao navegar pelas aplicações o sistema exibe as informações associadas.
Fluxo Alternativo	1. No passo 1 do fluxo principal, caso não haja informações sobre a aplicação, nada será exibido.

[RF – 12] - Configurar Gerenciador de Aplicações	
O *Gerenciador de Aplicações* deve prover uma interface de configuração de suas propriedades.	
Atores	Usuário
Prioridade	Desejável

Entradas e pré-condições	• Bloquear execução de aplicações *AUTOSTART*.
	• Bloquear notificação de novas aplicações.
	• Categorias:
	  o Especificar categoria inicial das aplicações
	  o Inserir/Remover categorias
	  o Adicionar/Mover aplicações às categorias
	• Exibir informações da aplicação durante a navegação.
	• Módulo Inteligente
	• Visualização de navegação em forma de:
	  o Lista
	  o Ícones

Saídas e pós-condições	• *Gerenciador de Aplicações* personalizado pelo usuário.
Fluxos de Eventos	
Fluxo Principal	1. Usuário acessa a interface do *Gerenciador de Aplicações*. 2. Usuário pressiona o botão de configuração. 3. Sistema exibe uma nova tela com os campos a serem preenchidos (vide entradas). 4. Usuário modifica os dados que desejar. 5. Sistema registra as modificações.

[RF – 13] – Obter Dados da Plataforma	
O *Gerenciador de Aplicações* deve prover acesso a determinados dados sobre a plataforma onde as aplicações serão executadas.	
Atores	Aplicação
Prioridade	Importante

Entradas e pré-condições	• Entradas:
	  o Aplicações existentes
	  o Aplicações em execução
	  o Aplicação atualmente em foco
	  o Aplicação visível
	  o Mapeamento (estado, aplicação)
	  o Prioridade da aplicação
	  o Tamanho da pilha de aplicações
	• A Aplicação solicitante deve ter permissão para obter os dados da plataforma

Saídas e pós-condições	• Dados solicitados são repassados a Aplicação solicitante
Fluxos de Eventos	
Fluxo Principal	1. Uma aplicação requisita ao *Gerenciador de Aplicações* dados referentes à plataforma de execução. 2. O *Gerenciador de Aplicações* checa a segurança de acessibilidade. 3. O *Gerenciador de Aplicações* repassa as informações à aplicação solicitante.
Fluxo Excepcional	1. No passo 2 do fluxo principal, o *Gerenciador de Aplicações* nega à aplicação acesso aos dados da plataforma. 2. Uma *SecurityException* é lançada. 3. Uma mensagem de erro é apresentada na tela (dispositivo de saída)

[RF – 14] – Obter Interação com Outra Aplicação	
O *Gerenciador de Aplicações* pode ser requisitado por uma aplicação a conceder um meio de comunicação entre ambas.	
Atores	Aplicação
Prioridade	Desejável
Entradas e pré-condições	• As aplicações devem existir no sistema (No futuro deverão existir na rede) • A aplicação solicitante deve ter permissão para solicitar interação com outra aplicação
Saídas e pós-condições	• O *Gerenciador de Aplicações* repassa as aplicações para o módulo de *Comunicação Inter-aplicações* do *middleware*.
Fluxos de Eventos	

Fluxo Principal	1. Uma aplicação A – em execução – requisita comunicação com uma aplicação existente B. 2. O *Gerenciador de Aplicações* checa a segurança de acessibilidade. 3. O *Gerenciador de Aplicações* permite a comunicação entre as aplicações. 4. O *Gerenciador de Aplicações* verifica estado da aplicação B – *STARTED*. 5. O *Gerenciador de Aplicações* repassa as aplicações ao módulo de Comunicação Inter-aplicações do *middleware*.
Fluxo Alternativo	1. No passo 4 do fluxo principal, o *Gerenciador de Aplicações* pode ser obrigado a carregar, iniciar ou executar a aplicação B, caso a mesma encontre-se em um estado diferente de *STARTED*.
Fluxo Excepcional	1. No passo 2 do fluxo principal, o *Gerenciador de Aplicações* verifica que a aplicação A não tem acessibilidade a iniciar comunicação com outra. 2. Uma *SecurtyException* é lançada. 3. Uma mensagem de erro é apresentada na tela (dispositivo de saída) do usuário.

[RF – 15] – Recuperar Aplicação	
O *Gerenciador de Aplicações* sob a configuração de "Módulo Inteligente" pode guardar aplicações pausadas em uma pilha, fornecendo ao usuário recuperação das aplicações através de uma interface similar à acessada no *Windows* através do pressionamento em combinação das teclas *ALT+ATB*.	
Atores	Usuário
Prioridade	Desejável
Entradas e pré-condições	• Configuração "Módulo Inteligente" deve estar ativa • Deve existir pelo menos uma aplicação na pilha.
Saídas e pós-condições	• O *Gerenciador de Aplicações* apresenta uma interface com as aplicações existentes na pilha. • Tendo apenas uma aplicação na pilha, ela será automaticamente apresentada na tela (dispositivo de saída) do sistema.
Fluxos de Eventos	
Fluxo Principal	1. O usuário pressiona uma combinação de teclas no controle remoto. 2. O *Gerenciador de Aplicações* captura o evento. 3. O *Gerenciador de Aplicações* verifica que há aplicações na pilha. 4. O *Gerenciador de Aplicações* inicia a aplicação interna de *Recuperação de Aplicações*.
Fluxo Alternativo	1. No passo 3 do fluxo principal, não há.aplicações na pilha, sendo ignorado os passos restantes.

[RF – 16] – Realizar Download de uma Aplicação
O *Gerenciador de Aplicações* ao acessar a *AIT* percebe que a aplicação está sinalizada como remota ou que sua versão não condiz com a do terminal, contudo na *AIT* está sinalizado a existência de outra para esta versão. Assim o sistema acessa a rede e realiza o *download* da aplicação.

Atores	*Gerenciador de Aplicações*
Prioridade	Desejável
Entradas e pré-condições	• Aplicação ser sinalizada na *AIT* como *REMOTE* • Versão da aplicação ser diferente da suportada no terminal
Saídas e pós-condições	• O *Gerenciador de Aplicações* carrega a aplicação do servidor remoto
Fluxos de Eventos	
Fluxo Principal	5. Aplicação é sinalizada na *AIT* como *REMOTE* 6. O *Gerenciador de Aplicações* requisita acesso à rede. 7. O *Gerenciador de Aplicações* acessa o *provider* da aplicação. 8. O *Gerenciador de Aplicações* carrega a aplicação no terminal de acesso.
Fluxo Alternativo	2. No passo 1 do fluxo principal, a.aplicações não é sinalizada como *REMOTE* estando multiplexada no fluxo. 3. O *Gerenciador de Aplicações* verifica o perfil da aplicação. 4. Não condiz com o perfil do terminal. 5. Retorna ao passo 2 do fluxo principal.

3. Requisistos Não-Funcionais

3.1. Requisitos de Sistema

[NF-01] - Sistema Operacionais Suportados	
O Gerenciador de Aplicações aqui apresentado necessita ser executado em sistemas operacionais dotados de uma máquina virtual Java *(JVM)*.	
Casos de Uso Associados	• *Todos*
Prioridade do Requisito	• Essencial

[NF-02] - Comunicação Inter-Aplicações	
O Gerenciador de Aplicações necessita de um módulo que provenha a Comunicação Inter-Aplicações, capacitando a interação entre as aplicações.	
Casos de Uso Associados	• *[RF – 14] – Obter Interação com Outra Aplicação*
Prioridade do Requisito	• Essencial

[NF-03] - Inicialização Automática	
No momento que o terminal de acesso for ligado, o Gerenciador de Aplicações deve ser executado automaticamente. Ele será executado como um *Thread* do sistema de baixíssima prioridade.	
Casos de Uso Associados	• *Todos*
Prioridade do Requisito	• Essencial

[NF-04] - Processamento de Dados	
O Gerenciador de Aplicações fará uso do módulo de Processamento de Dados para a obtenção das aplicações (carrossel). Este módulo do *middleware* registrará o Gerenciador de Aplicações como um *listener* da notificação de chegada de novas aplicações.	
Casos de Uso Associados	• *[RF - 07] – Receber Notificação de Download de Aplicação*
Prioridade do Requisito	• Essencial

3.2. Requisitos de Ergonomia

[NF-05] – Controle Remoto Ergonômico	
O controle remoto deve possuir um design capaz de trazer conforto ao usuário de TV Digital Interativa, ou seja, ter formato, dimensões e disposição dos botões que se adeqüem melhor à mão humana.	
Casos de Uso Associados	• *[RF – 03] – Executar uma Aplicação* • *[RF – 04] – Parar uma Aplicação* • *[RF – 06] – Limpar Tela (CLEAR)* • *[RF – 08] – Salvar uma Aplicação* • *[RF – 09] – Excluir uma Aplicação* • *[RF – 10] – Navegar nas Aplicações* • *[RF – 12] – Configurar Gerenciador de Aplicações*
Prioridade do Requisito	• Desejável

3.3. Requisitos de Desempenho

[NF-06] – Carga gráfica não deve comprometer o funcionamento das aplicações Internas	
A interface gráfica das aplicações internas não deve comprometer o funcionamento das mesmas.	
Casos de Uso Associados	• *[RF – 03] – Executar uma Aplicação* • *[RF – 08] – Salvar uma Aplicação* • *[RF – 09] – Excluir uma Aplicação* • *[RF – 10] – Navegar nas Aplicações* • *[RF – 12] – Configurar Gerenciador de Aplicações*
Prioridade do Requisito	• Importante

[NF-07] – Tamanho do Gerenciador de Aplicações	
O módulo deve ocupar pouco espaço no sistema de armazenamento. O tamanho que o Gerenciador de Aplicações e suas aplicações internas ocuparão na memória não deve exceder 500 Kilobytes.	
Casos de Uso Associados	• *Todos*
Prioridade do Requisito	• Desejável

[NF-08] – Tempo de Resposta	
A execução de alguma operação solicitada ao Gerenciador de Aplicações não deve exceder 100 milissegundos.	
Casos de Uso Associados	• *Todos*
Prioridade do Requisito	• Essencial

3.4. Requisitos de Confiabilidade

[NF-09] – Integridade das Inforamções	
A integridade das informações da AIT mantida pelo Gerenciador de Aplicações deve ser preservada.	
Casos de Uso Associados	• *[RF – 08] – Salvar uma Aplicação* • *[RF – 09] – Excluir uma Aplicação* • *[RF – 11] – Visualizar Informações da Aplicação* • *[RF – 12] - Configurar Gerenciador de Aplicações* • *[RF – 13] – Obter Dados da Plataforma*
Prioridade do Requisito	• Essencial

3.5. Requisitos de Segurança

[NF-10] – Execução de Processos Segura	
Os processos executados pelo Gerenciador de Aplicações não devem interferir no funcionamento das demais aplicações do sistema.	
Casos de Uso Associados	• *Todos*
Prioridade do Requisito	• Essencial

[NF-11] – Controle de Permissões	
As aplicações poderão carregar, inicializar, executar, parar e pausar outras aplicações, bem como obter dados da plataforma e interação com outras aplicações, além de concorrer pelos recursos compartilhados do sistema. Isso só ocorre se as mesmas possuírem permissões suficientes para isso.	
Casos de Uso Associados	• *[RF – 01] - Carregar uma Aplicação* • *[RF – 02] – Inicializar uma Aplicação* • *[RF – 03] – Executar uma Aplicação* • *[RF – 04] – Parar uma Aplicação* • *[RF – 05] – Pausar uma Aplicação* • *[RF – 13] – Obter Dados da Plataforma* • *[RF – 14] – Obter Interação com Outra Aplicação*
Prioridade do Requisito	• Essencial

3.6. Requisitos de Robustez

[NF-12] – Tolerância a Falhas
O Gerenciador de Aplicações deve continuar em execução de maneira confiável mesmo após a

ocorrência de alguns erros internos.	
Casos de Uso Associados	• *Todos*
Prioridade do Requisito	• Essencial

[NF-13] – *Watchdog Timer*	
O *Watchdog Timer* funcionaria como um cão de guarda do *Gerenciador de Aplicações*, monitorando sua execução em busca de eventuais falhas. Se o *Gerenciador de Aplicações* parar de responder ao *WTD* por um determinado período de tempo, o sistema é reiniciado. O *Watchdog Timer* também é aplicado às aplicações do sistema. Como a mesma é inserida dentro de uma *Thread*, inicializada pelo *Gerenciador de Aplicações*, compete a *Thread* informar de sua correta execução ao *WTD* dentro do prazo definido. Caso isso não ocorra a aplicação vinculada à *Thread* será finalizada. Devido a sua complexidade há possibilidades reais de que *Gerenciador de Aplicações* entre em *deadlock* ou *livelock*. O *WTD* trabalha como um plano de contingência.	
Casos de Uso Associados	• Todos
Prioridade do Requisito	• Essencial

3.7. Requisitos de Usabilidade

[NF-14] – Interface das Aplicações Internas Deve Ser Intuitiva	
As aplicações internas devem ser de simples utilização, contando com o auxílio de ícones e figuras explicativas. Desta maneira, tende-se a tornar o sistema mais acessível às pessoas de diferentes níveis de familiaridade com informática.	
Casos de Uso Associados	• *[RF – 03] – Executar uma Aplicação* • *[RF – 08] – Salvar uma Aplicação* • *[RF – 09] – Excluir uma Aplicação* • *[RF – 10] – Navegar nas Aplicações* • *[RF – 12] – Configurar Gerenciador de Aplicações*
Prioridade do Requisito	• Importante

[NF-15] – Ajuda	
As aplicações internas devem permitir que o usuário acesse a ajuda a qualquer momento da execução. A ajuda deve instruir o usuário a interagir com o sistema através de textos explicativos sobre as funcionalidades do mesmo. A aplicação também deverá prover um mecanismo de ajuda em áudio, para indivíduos analfabetos ou com dificuldades de leitura. É ainda interessante que haja tutoriais passo a passo para a execução de cada tarefa.	
Casos de Uso Associados	• *[RF – 03] – Executar uma Aplicação* • *[RF – 08] – Salvar uma Aplicação* • *[RF – 09] – Excluir uma Aplicação* • *[RF – 10] – Navegar nas Aplicações* • *[RF – 12] – Configurar Gerenciador de Aplicações*
Prioridade do Requisito	• Desejável

Apêndice IV

Assinaturas Dos Métodos

Apresento neste apêncide a assinatura dos métodos das principais classes implementadas durante este trabalho de graduação. Não será possível apresentar todo o código, nem todas as classes o que tornaria este trabalho impraticável.

ApplicationManager

```
package gerenciador.controlador;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.MalformedURLException;
import java.util.Iterator;
import java.util.Stack;
import java.util.Vector;

import javax.tv.locator.Locator;
import javax.tv.locator.MalformedLocatorException;
import javax.tv.xlet.Xlet;
import javax.tv.xlet.XletContext;
import javax.tv.xlet.XletStateChangeException;
import javax.xml.parsers.ParserConfigurationException;

import org.dvb.application.AppID;
import org.dvb.application.AppProxy;
import org.xml.sax.SAXException;

public class ApplicationManager implements Runnable {


      private static int currentAppId = 0;
      private static ApplicationManager instance = null;

      public synchronized static ApplicationManager getInstance() {}

      // Atributos
      private AIT ait;

      private Watchdog watchdog;

      private UserEventManager userEventManager;

      private ComunicatorManager comunicatorManager;

      private StorageManager storageManager;
```

```
        // Construtor
        private ApplicationManager()

        // Métodos
        public void notifyApplicationDownload(Locator locator)

        private void initXlet(XletProxy xletProxy, XletContext context)

        private XletProxy loadXlet(XletDescriptor descriptor)

        public boolean load(AppProxy proxy)

        public boolean init(AppProxy proxy, XletContext context)

        public void initNative(NativeContext context)

        public boolean pause(AppProxy proxy)

        public boolean start(AppProxy proxy)

        public boolean start(AppProxy proxy, String[] args)

        private void startXlet(XletProxy proxy, Xlet xlet)

        private void startXlet(XletProxy proxy, Xlet xlet, String[] args)

        public void clear()

        public void removeXlet(Locator locator)

        public boolean stop(AppProxy application, boolean forced)

        public void startNative(int appType)

        protected static void restart()

        public synchronized void run()

        public UserEventManager getUserEventManager()

        public ComunicatorManager getComunicatorManager()

        public StorageManager getStorageManager()

        public AIT getAit()
}
```

AIT

```
package gerenciador.controlador;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Vector;
```

```
import javax.tv.locator.Locator;
import javax.tv.locator.MalformedLocatorException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.ParserConfigurationException;

import org.apache.crimson.jaxp.DocumentBuilderFactoryImpl;
import org.dvb.application.AppID;
import org.dvb.application.AppProxy;
import org.w3c.dom.DOMException;
import org.xml.sax.SAXException;

public class AIT {

      private static AIT instance = null;

      // protected
      public static AIT getInstance()
            throws MalformedLocatorException,ParserConfigurationException,
            SAXException, IOException

      // > Mapeamento das Aplicacoes Nativas: Integer X
      // NativeApplicationProxy
      private Hashtable nativeProxies;

      // > Mapeamento dos Xlets: AppId X XletProxy
      private Hashtable xletProxies;

      // > Aplicacoes atualmente em visibilidade
      private Vector visibleApplications;

      // > Xlets atualmente sendo executados: AppID X Thread
      private Hashtable runningXlets;

      // > Aplicacoes residentes atualmente sendo executadas
      private Vector runningNativeProxies;

      /**
       * Busca no arquivo de inicializacao do Application Manager as
       * aplicacoes existentes. Sejam Xlets ou Nativas
       *
       * @throws MalformedLocatorException
       * @throws ParserConfigurationException
       * @throws IOException
       * @throws SAXException
       */
      private AIT()

      public void clear()

      private void fillResidents()

      public Iterator getNativeProxies()

      public NativeProxy getNativeProxy(int kind)

      public Iterator getRunningNative()

      public Vector getVisibleApplications()

      public Iterator getXletProxies()
```

```
      public XletProxy getXletProxy(AppID id)

      public Thread getXletRunning(AppID id)

      public Iterator getXletsRunning()

      public boolean hasVisibleApplications()

      private void putNative(Integer type, NativeProxy proxy)

      public void putRunningNativeProxy(AppProxy application)

      public void putRunningXlet
            (AppID identifier, Thread executingApplication)

      public void putVisibleApplication(AppProxy app)

      public void putXletProxy(AppID appId, XletProxy proxy)

      public void removeRunningApplication(AppProxy application)

      public void removeVisibleApplication(AppProxy app)

      public void removeXlet(Locator locator)

      public void removeXlet(XletProxy proxy)

      private void setMappings(Hashtable applications)

      public void saveApplications()
}
```

AppProxy

```
package org.dvb.application;
/**
 * AppProxy.java
 * Um objeto AppProxy e um proxy para uma aplicacao.
 *
 * Um objeto AppProxy e um proxy para uma aplicacao. Uma chamada
 * de execucao, parada ou pausa acarretara em respectivas chamadas
 * de execucao, parada ou pausa ao Application Manager residente
 * no middleware, o qual repassara estas chamadas a aplicacao
 * vinculada a este objeto AppProxy. Cada um destes tres metodos
 * pode lancar Security Exception, caso a aplicacao chamadora
 * nao esteja capacitada a fazer isso.
 *
 * Estas chamadas são assíncronas e resultarãoo sempre em
 * um AppStateChangedEvent mesmo se a chamada de método não
 * obter exito. Caso a chamada de metodo nao
 * obter exito, qualquer chamada ao metodo hasFailed do AppStateChangeEvent
 * correspondente retornara true.
 *
 * Alguns metodos permitem ao AppProxy transitar entre varios estados
 * antes que o estado final sja alcancado. Caso esta composicao de
 * transicoes de estados nao obtenha exito em qualquer ponto, o
 * AppStateChangedEvent resultante deve conter um fromstate, o qual
 * representa o ultimo estado na transicao no qual o AppProxy obteve
 * exito em alcancar e um toState, o qual deve representar o estado que o
 * mesmo deveria ter alcancado caso a transicao houvesse alcancado exito.
 */
```

```
public interface AppProxy {

      //> A aplicacao esta executando. */
    public static int STARTED = 0;
      //> A aplicacao esta parada. */
    public static int STOPPED = 1;

    public void addAppStateChangeEventListener
            (AppStateChangeEventListener listener);

    public int getState();

    public void pause() throws SecurityException;

    public void removeAppStateChangeEventListener
            (AppStateChangeEventListener listener);

    public void resume() throws SecurityException;

    public void start() throws SecurityException;

    public void start(String[] args) throws SecurityException;

    public void stop( boolean forced ) throws SecurityException;
}
```

AppAttributes

```
package org.dvb.application;

/**
 * AppAttributes.java
 * Aqui estao declarados os metodos definidos para a interface
 * AppAtribute.
 *
 * A Interface AppAttributes e uma mapeamento de varias
 * informacoes sobre umaaplicacao registrada. Para aplicacoes
 * que estao sinalizadas na AIT, o mapeamento entre os valores
 * retornados pelos metodos desta classe, os campos e os
 * descritores da AIT devem ser especificadas no corpo
 * desta especificação.
 *
 * Instâncias de objetos que implementem esta interface sao
 * imutaveis e populados antes da mesma ser retornada para
 * uma aplicacao.
 */
public abstract class AppAttributes {

      /** Tipos das aplicações */
      // Aplicações nativas
      public static final int NATIVE_APPLICATIONS = 0;
      // Aplicações Xlets
      public static final int XLET_APPLICATIONS = 1;

      /** Prioridades das Aplicações */
      public static final int TOP_PRIORITY = 100;
      public static final int MAX_PRIORITY = 75;
      public static final int MEDIUM_PRIORITY = 50;
      public static final int LOW_PRIORITY = 25;
      public static final int MINIMUN_PRIORITY = 0;
```

```
        //> Vinculacao a servico.
        private boolean serviceBound;
        //> Nome da Aplicacao
        private String name;
        //> Prioridade da Aplicacao
        private int priority;
        //> Tipo da Aplicacao
        private int type;
        //> Visibilidade da Aplicacao
        private boolean visible;

        public AppAttributes()

        public AppAttributes(boolean bound, String name, int priority)

        public boolean isServiceBound()

        public String getName()

        public int getPriority()

        public abstract Object getProperty(String index);

        public abstract int getType();

        public void setName(String name)

        protected void setPriority(int priority)

        public void setServiceBound(boolean serviceBound)

        public void setType(int type)

        public boolean isVisible()

        public void setVisible(boolean visible)
}
```

XLET

```
package javax.tv.xlet;
/**
 * Xlet.java Esta interface permite um que gerenciador de aplicacoes crie,
 * incialize, pause e destrua um Xlet.
 *
 * Um Xlet e uma aplicacao ou servico projetado para ser executado
 * e controlado por um gerenciador de aplicacoes via sua interface
 * de ciclo de vida. Os estados do ciclo de vida permitem que o
 * gerenciador de aplicacoes gerencie as atividades de multiplos
 * Xlets dentro de um ambiente de execucao, selecionando quais Xlets
 * estao ativos em um dado momento. O gerenciador de aplicacoes mantem
 * o estado do Xlet e invoca metodos do Xlet via os metodos de controle
 * do ciclo de vida O Xlet implmenta estes metodos para atualizar suas
 * atividades internas e uso de recursos de acordo com o gerenciador de
 * aplicacoes. O Xlet pode iniciar algumas mudancas de estado por conta
 * propria, tendo que informa-las ao gerenciador de aplicacoes via os
 * metodos presentes no XletContext.
 *
 * Com o objetivo de prover interoperabilidade entre Xlets e gerenciadores
 * de aplicacoes, todas as classes Xlet devem possuir um contrutor
 * publico sem argumentos.
```

```
 */
public interface Xlet {

      public void destroyXlet(boolean unconditional)
                  throws XletStateChangeException;

      public void initXlet(XletContext ctx)
                  throws XletStateChangeException;

      public void pauseXlet()
                  throws XletStateChangeException;

      public void startXlet() throws XletStateChangeException;
}
```

XLETCONTEXT

```
package javax.tv.xlet;
/**
 * XletContext.java
 * Interface que prove metodos que permitem que um Xlet descubra
 *  informacoes sobre o seu ambiente.
 *
 * Um objeto do tipo XletContext e passado aos Xlets quando eles sao
 * incializados.
 * Ele possibilita que um Xlet recupere propriedades, assim como um meio de
 * sinalizar mudancas no estado interno.
 */
import java.util.Iterator;


public interface XletContext {
      public static final int APPLICATIONS = 0;
      public static final int APPLICATION_STATES = 1;
      public static final int RUNNING_APPLICATIONS = 2;
      public static final int NATIVES = 3;

      public static final String ARGS = "args";

      public Object getXletProperty( String key );

      public Iterator getPlataformData( int key );

      public void notifyDestroyed();

      public void resumeRequest();

      public Iterator getProxysInStack();
}
```

XAIT

```
package gerenciador.controlador.cicloVida.xlet;

import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;

import javax.tv.locator.Locator;
```

```
import org.dvb.application.AppID;
import org.dvb.application.AppIcon;
import org.dvb.application.LanguageNotAvailableException;

/**
 * Classe que armazena as informações necessárias do Xlet.
 *
 * E uma Information Table reduzida, unica para cada Xlet.
 * Aqui pode-se encontrar toda os descritores, atributos,
 * parametros, propriedades, bem como  o identificador
 * do Xlet.
 */
public class XAIT {

    /** ATRIBUTOS */
    //> O Id da Xlet.
    private AppID xletId;

    //> O Loader do Xlet
    private XletLoader xletLoader;

    //> Os atributos e descritores do Xlet
    private XletDescriptor xletDescriptor;

    /**
     * @brief Construtor da XAIT
     *
     * @param xletId
     * @param xletLoader
     * @param xletWrapper
     * @param xletDescriptor
     * @throws IOException
     */
    public XAIT(AppID xletId, XletDescriptor descriptor)
      throws IOException

    public XletDescriptor getXletDescriptor()

    public void setXletDescriptor(XletDescriptor xletDesc)

    public AppID getIdentifier()

    public void setXletId(AppID xletId)

    public XletLoader getXletLoader()

    public void setXletLoader(boolean set)
      throws MalformedURLException

    public String getName()

    public int getPriority()

    public boolean isServiceBound()

    public boolean isVisible()

    public void setVisible(boolean visible)

    public int getDataSize()
```

```
    public AppIcon getIcon()

    public String getInfo()

    public String getName(String iso639code)
        throws LanguageNotAvailableException

      public String[] getNames()

    public int getOid()

    public ArrayList getProfiles()

    public Object getProperty(String index)

    public Locator getServiceLocator()

    public int getSize()

    public int getType()

    public ArrayList getVersions(String versions)

    public String getXletVendor()

    public boolean isStartable()

    public void putName(String iso639code, String name)

    public String getMainClassName()
      throws IOException

    public Class loadClass(String name)
      throws ClassNotFoundException

    public Class loadMainClass()
      throws ClassNotFoundException

    public String getXletBaseDir()

    public String getXletClasspath()

    public String getXletJarFile()

    public ArrayList getXletParameters()

    public ArrayList getLanguages()

    public int getAID()

    public int getOID()
}
```

NATIVE

```
package gerenciador.controlador.cicloVida.native;

import java.io.IOException;
```

```
public abstract class Native {

    private Process applicationProcess;
.
    private String executableName;

    private NativeContext context;

    public Native(String executableName)

    public void init(NativeContext context)

    public void start() throws SecurityException

    public void start(String[] args) throws SecurityException

    public void stop(boolean forced) throws SecurityException

    public abstract void pause();

}
```

NativeContext

```
package gerenciador.controlador.cicloVida.native;

public interface NativeContext {

    public void notifyDestroy();

    public void notifyPaused();

    public void resumeRequest();
}
```

ApplicationManagerScanner

```
package gerenciador.processamento;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Hashtable;

import javax.tv.locator.Locator;
import javax.tv.locator.LocatorFactoryImpl;
import javax.tv.locator.MalformedLocatorException;

import org.dvb.application.AppID;
import org.w3c.dom.DOMException;

import gerenciador.controlador.cicloVida.xlet.XAIT;
import gerenciador.controlador.cicloVida.xlet.XletDescriptor;
import gerenciador.controlador.cicloVida.xlet.XletProxy;

public class ApplicationManagerScanner {

    org.w3c.dom.Document document;
```

```
    public ApplicationManagerScanner(org.w3c.dom.Document document)

    public Hashtable visitDocument()
            throws DOMException, FileNotFoundException,
            MalformedLocatorException, IOException

    Locator visitElement_xlet_locator(org.w3c.dom.Element element)

    String visitElement_xlet_oid(org.w3c.dom.Element element)

    String visitElement_xlet_aid(org.w3c.dom.Element element)

    AppID visitElement_xlet_appID(org.w3c.dom.Element element)

    XAIT visitElement_xlet(org.w3c.dom.Element element)
            throws NumberFormatException, FileNotFoundException,
                   IOException, MalformedLocatorException

    ArrayList visitElement_xlets(org.w3c.dom.Element element)
            throws NumberFormatException, FileNotFoundException,
                   IOException, MalformedLocatorException


    Hashtable visitElement_applications(org.w3c.dom.Element element)
            throws NumberFormatException, FileNotFoundException,
                   IOException, MalformedLocatorException
}
```

APPLICATIONMANAGERWRITER

```
package gerenciador.processamento;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.util.Iterator;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.CDATASection;
import org.w3c.dom.Comment;
import org.w3c.dom.Document;
import org.w3c.dom.DocumentType;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.ProcessingInstruction;
import org.w3c.dom.Text;

import gerenciador.controlador.cicloVida.xlet.XletProxy;
import gerenciador.controlador.AIT;

public class ApplicationManagerWriter {

      private PrintWriter out;

      private org.w3c.dom.Document document;
```

```
        public ApplicationManagerWriter(File output)

        private void close()

        private void write(Node node)

        private void write(Node node, String indent)

        private String fixup(String s)

        public void toXML()

        public void toXML(AIT ait)
}
```